Lab 4 - BCC406/PCC177

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Uso de Framework (TensorFlow) e K-Fold

Prof. Eduardo e Prof. Pedro

Objetivos:

- Classificação utilizando TensorFlow.
- Regressão Logística.
- Cálculos de métircas

Data da entrega : 04/02

- Complete o código (marcado com 'ToDo') e quando requisitado, escreva textos diretamente nos notebooks. Onde tiver *None*, substitua pelo seu código.
- Execute todo notebook e salve tudo em um PDF **nomeado** como "NomeSobrenome-LabX.pdf"
- Envie o PDF via google FORM
- Envie o *.ipynb* também.

Preparação do ambiente e Tratamento dos dados

Preparação do ambiente

Importação das bibliotecas

Primeiro precisamos importar os pacotes. Vamos executar a célula abaixo para importar todos os pacotes que precisaremos.

- *TensorFlow* é o pacote fundamental de operações de *Deep Learning*.
- *numpy* é o pacote fundamental para a computação científica com Python.
- *h5py* é um pacote comum para interagir com um conjunto de dados armazenado em um arquivo H5.
- *matplotlib* é uma biblioteca famosa para plotar gráficos em Python.
- *PIL* e *scipy* são usados aqui para carregar as imagens e testar seu modelo final.
- *Scikit Learn* é um pacote muito utilizado para treinamento de modelos e outros algoritmos de *machine learning*.

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy

from sklearn.metrics import accuracy_score
import tensorflow as tf

from tensorflow import keras
```

Configurando os *plots* de gráficos

O próximo passo é configurar o *matplotlib* e a geração de valores aleatórios.

In [14]:
```
%matplotlib inline
plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload
%autoreload 2

np.random.seed(1)
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Configurando o Google Colab.

Configurando o Google Colab para acessar os nossos dados.

In [15]:
```
# Você vai precisar fazer o upload dos arquivos no seu drive (faer na pasta raiz
# não se esqueça de ajustar o path para o seu drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
```

Carregando e préprocessamento dos dados

In [16]:
```
# Função para ler os dados (gato/não-gato)
def load_dataset():
    def _load_data():
      train_dataset = h5py.File('/content/drive/MyDrive/treino_teste/train_catvn
      train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train
      train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train

      test_dataset = h5py.File('/content/drive/MyDrive/treino_teste/test_catvnon
      test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set
      test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set

      classes = np.array(test_dataset["list_classes"][:]) # the list of classes
      train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
      test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))
```

```
    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_ori

  def _preprocess_dataset(_treino_x_orig, _teste_x_orig):
    # Formate o conjunto de treinamento e teste dados de treinamento e teste p
    # de tamanho (num_px, num_px, 3) sejam vetores de forma (num_px * num_px *
    print(_treino_x_orig.shape)
    print(_teste_x_orig.shape)
    #_treino_x_orig = _treino_x_orig.reshape(_treino_x_orig.shape[0], 64 * 64
    #_teste_x_orig = _teste_x_orig.reshape(_teste_x_orig.shape[0], 64 * 64 * 3
    _treino_x_vet = _treino_x_orig.reshape(_treino_x_orig.shape[0], -1) # ToDo
    _teste_x_vet = _teste_x_orig.reshape(_teste_x_orig.shape[0], -1) # ToDo: v

    # Normalize os dados (colocar no intervalo [0.0, 1.0])
    _treino_x = _treino_x_vet/255. # ToDo: normalize os dados de treinamento a
    _teste_x = _teste_x_vet/255. # ToDo: normalize os dados de teste aqui
    return _treino_x, _teste_x

  treino_x_orig, treino_y, teste_x_orig, teste_y, classes = _load_data()
  treino_x, teste_x = _preprocess_dataset(treino_x_orig, teste_x_orig)
  return treino_x, treino_y, teste_x, teste_y, classes
```

In [17]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
```

Carregando os dados

In [18]:
```
# Lendo os dados (gato/não-gato)
treino_x, treino_y, teste_x, teste_y, classes = load_dataset()
```

```
(209, 64, 64, 3)
(50, 64, 64, 3)
```

Treinamento do modelo (100pt)

Há diversos frameworks para criação de modelos de *deep learning*, como TensorFlow e PyTorch. Nesta prática, usaremos o TensorFlow.

Função para treinar um modelo

A primeira parte envolve a criação de uma função que será usada para treinar os próximos modelos. Essa função será usada em todos os modelos testados.

In [19]:
```
def treinar_modelo(modelo, treino_x, treino_y, epochs=100):
    # Setando a seed
    np.random.seed(1)

    # Compilando o modelo
    modelo.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy']
                    )

    # Imprimindo a arquitetura da rede proposta
```

```
    modelo.summary()

    # Treinando o modelo
    modelo.fit(treino_x, treino_y.reshape(-1), epochs=epochs)
    return modelo
```

Modelo 1: Testando um modelo com uma camada oculta com 8 neurônios (10pt)

Definição de um modelo com uma camada oculta (8 neurônios) e uma camada de saída com um neurônio (gato e não gato). Usaremos a ativação ReLU (*Retified Linear Unity*) na camada oculta e a *sigmoid* na camada de saída. Para classificação de classes 0 ou 1, pode-se ter um único neurônio de saída e deve-se usar a operação sigmoid antes de se calcular o custo (mean-squared error ou binary cross entropy).

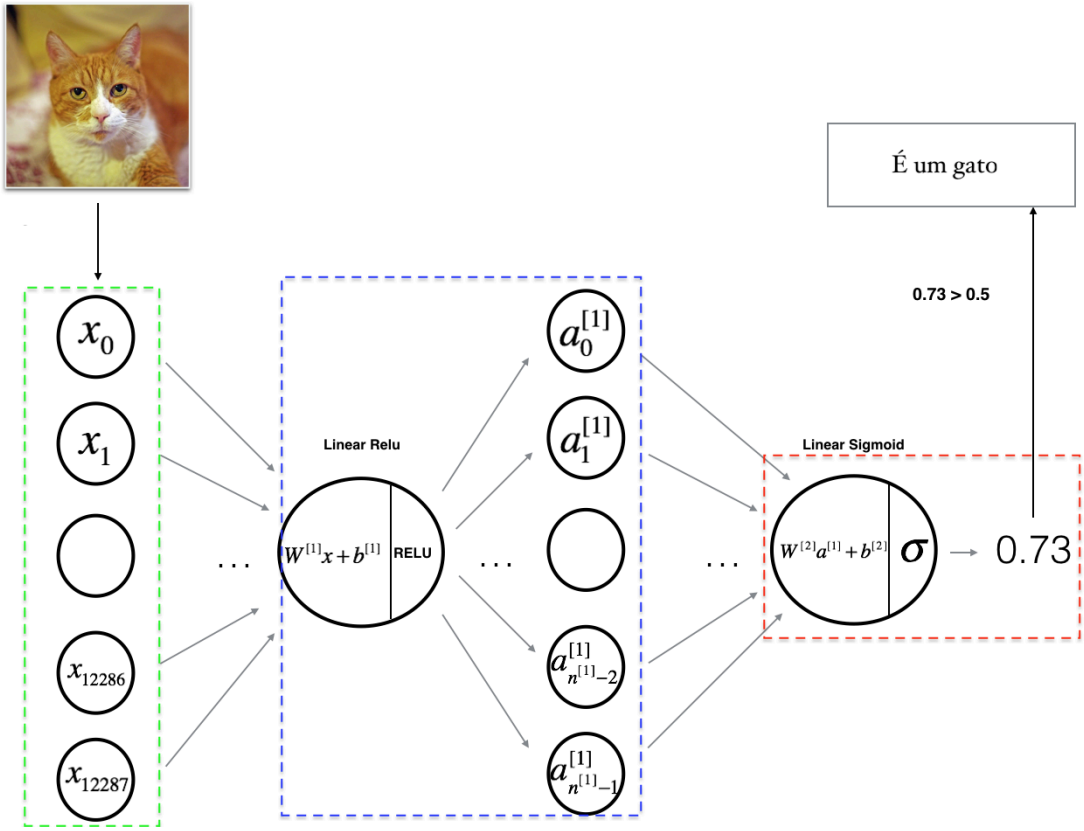

Figura 1: Rede neural com 2 camadas.
Resumo do modelo: ***ENTRADA -> LINEAR -> RELU -> LINEAR -> SIGMOID -> SAIDA***.

Definição do modelo (5pt)

A primeira etapa é a definição da arquitetura do modelo. Para este primeiro modelo será usado um modelo com somente oito neurônios.

In [20]:
```
# Definição do modelo
def modelo_1():
  _model = tf.keras.models.Sequential() # Crie um modelo sequencial com keras.Se
  _model.add(tf.keras.layers.Dense(8,activation = 'relu')) # ToDo: Adicione uma
```

```
  _model.add(tf.keras.layers.Dense(1,activation = 'sigmoid')) # ToDo: Adicione u
  return _model
```

Instanciando o modelo e testando (5pt)

Treine o modelo e depois **use os parâmetros treinados** para classificar as imagens de treinamento e teste e verificar a acurácia.

In [21]:
```
np.random.seed(1)
tf.random.set_seed(1)

# Criando o modelo
m1 = modelo_1() # ToDo: chame a função que define o modelo
# Treinando o modelo
m1 = treinar_modelo(m1, treino_x, treino_y) # ToDo: Chame a função para treinar

previsoes_treino = np.round(m1.predict(treino_x))
previsoes_test = np.round(m1.predict(teste_x))

## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_score(treino_y.reshape(-1), previsoes_
                                        # **dica** use o model.predict para pre
print(f'Acurácia no teste: {accuracy_score(teste_y.reshape(-1), previsoes_test)}
                                    # **dica** use o model.predict para predizer
```

Model: "sequential"

Layer (type)	Output Shape	
dense (Dense)	?	
dense_1 (Dense)	?	

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.5485 - loss: 1.0090
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4043 - loss: 0.6932
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6926
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6919
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6910
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6901
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6891
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6882
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6873
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6864
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6855
Epoch 12/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6846
Epoch 13/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6838
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6829
Epoch 15/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6821
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6812
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6804
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6796
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6788
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6781
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6773
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6765
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6758
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6751
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6744
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6737
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6730
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6609 - loss: 0.6723
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6716
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6609 - loss: 0.6710
```

```
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6703
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6609 - loss: 0.6697
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6691
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6609 - loss: 0.6685
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6679
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6673
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6667
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6661
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6609 - loss: 0.6656
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6650
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6609 - loss: 0.6645
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6640
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6635
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6609 - loss: 0.6630
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6625
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6609 - loss: 0.6620
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6615
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6609 - loss: 0.6610
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6606
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6601
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6609 - loss: 0.6597
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6609 - loss: 0.6592
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6588
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6584
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6580
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6576
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6572
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6568
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6564
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6561
```

```
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6557
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6553
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6550
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6546
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6543
Epoch 66/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6540
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6609 - loss: 0.6537
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6534
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6530
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6527
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6525
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6522
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6519
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6516
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6609 - loss: 0.6513
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6511
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6508
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6506
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6503
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6501
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6498
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6496
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6494
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6609 - loss: 0.6492
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6489
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6487
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6485
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6483
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6481
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6609 - loss: 0.6479
```

```
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6477
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6609 - loss: 0.6476
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6609 - loss: 0.6474
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6472
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6470
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6469
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6467
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6609 - loss: 0.6465
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6609 - loss: 0.6464
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6609 - loss: 0.6462
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step


Acurácia no treino: 0.6555023923444976
Acurácia no teste: 0.34
```

Resultado esperado: (pode ser diferente)

```
Acurácia treino = 81.34%
Acurácia teste = 52.00%
```

Modelo 2: Testando um modelo com uma camada oculta com 256 neurônios (15pt)

Definição do modelo (10pt)

In [22]:
```
# Definição do modelo
def modelo_2():
  _model = tf.keras.models.Sequential() # Crie um modelo sequencial com keras.Se
  _model.add(tf.keras.layers.Dense(256,activation = 'relu')) # ToDo: Adicione um
  _model.add(tf.keras.layers.Dense(1,activation = 'sigmoid')) # ToDo: Adicione u
  return _model
```

Crie um modelo com uma camada oculta (256 neurônios e ativação ReLu) e a camada de saída com um neurônio (ativação sigmoid).

Agora treine e teste o seu modelo.

In [23]:
```
np.random.seed(1)
tf.random.set_seed(1)

# Criando o modelo
m2 = modelo_2() # ToDo: chame a função que define o modelo
# Treinando o modelo
```

```
m2 = treinar_modelo(m2, treino_x, treino_y) # ToDo: Chame a função para treinar

previsoes_treino = np.round(m2.predict(treino_x))
previsoes_test = np.round(m2.predict(teste_x))

## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_score(treino_y.reshape(-1), previsoes_
                                        # **dica** use o model.predict para pre
print(f'Acurácia no teste: {accuracy_score(teste_y.reshape(-1), previsoes_test)}
                                    # **dica** use o model.predict para predizer
```

Model: "sequential_1"

Layer (type)	Output Shape	
dense_2 (Dense)	?	
dense_3 (Dense)	?	

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.6173 - loss: 4.8643
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.6609 - loss: 2.4995
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.5578 - loss: 1.0149
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.6717 - loss: 1.0337
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.6154 - loss: 0.6719
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 86ms/step - accuracy: 0.6700 - loss: 0.7114
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.5728 - loss: 0.6617
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 89ms/step - accuracy: 0.6945 - loss: 0.5489
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 89ms/step - accuracy: 0.7020 - loss: 0.5595
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step - accuracy: 0.7299 - loss: 0.5291
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 0.7755 - loss: 0.4967
Epoch 12/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 74ms/step - accuracy: 0.7671 - loss: 0.4994
Epoch 13/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 0.7942 - loss: 0.4866
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.7756 - loss: 0.4709
Epoch 15/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.8106 - loss: 0.4620
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8070 - loss: 0.4537
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.8057 - loss: 0.4436
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.8263 - loss: 0.4329
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.8211 - loss: 0.4229
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8280 - loss: 0.4117
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.8353 - loss: 0.4076
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.8442 - loss: 0.3971
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 0.8492 - loss: 0.3879
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.8389 - loss: 0.3854
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.8719 - loss: 0.3758
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.8690 - loss: 0.3677
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.8641 - loss: 0.3586
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.8635 - loss: 0.3570
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.9006 - loss: 0.3451
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.8974 - loss: 0.3322
```

```
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9043 - loss: 0.3248
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.9062 - loss: 0.3187
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.9062 - loss: 0.3107
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 0.9205 - loss: 0.3039
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.9205 - loss: 0.2984
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.9205 - loss: 0.2914
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - accuracy: 0.9241 - loss: 0.2853
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 76ms/step - accuracy: 0.9253 - loss: 0.2802
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9361 - loss: 0.2715
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 86ms/step - accuracy: 0.9482 - loss: 0.2631
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.9482 - loss: 0.2551
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9482 - loss: 0.2455
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.9535 - loss: 0.2359
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.9553 - loss: 0.2279
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9589 - loss: 0.2192
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.9589 - loss: 0.2116
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.9615 - loss: 0.2042
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 105ms/step - accuracy: 0.9742 - loss: 0.1968
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 74ms/step - accuracy: 0.9742 - loss: 0.1900
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - accuracy: 0.9837 - loss: 0.1830
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 124ms/step - accuracy: 0.9837 - loss: 0.1771
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.9837 - loss: 0.1716
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.9837 - loss: 0.1663
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.9837 - loss: 0.1614
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9837 - loss: 0.1571
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9837 - loss: 0.1520
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9837 - loss: 0.1485
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.9837 - loss: 0.1447
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.9837 - loss: 0.1436
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9768 - loss: 0.1434
```

```
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - accuracy: 0.9837 - loss: 0.1445
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9837 - loss: 0.1477
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.9784 - loss: 0.1500
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.9765 - loss: 0.1558
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.9739 - loss: 0.1599
Epoch 66/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 78ms/step - accuracy: 0.9644 - loss: 0.1626
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.9626 - loss: 0.1693
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9660 - loss: 0.1710
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9712 - loss: 0.1538
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.9826 - loss: 0.1449
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - accuracy: 0.9779 - loss: 0.1288
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.9441 - loss: 0.1551
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.8533 - loss: 0.2768
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.7966 - loss: 0.4229
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.8197 - loss: 0.3856
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.8990 - loss: 0.2220
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.9183 - loss: 0.1777
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8872 - loss: 0.2479
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.9322 - loss: 0.2028
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - accuracy: 0.9787 - loss: 0.1045
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9797 - loss: 0.0994
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.9931 - loss: 0.0715
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9931 - loss: 0.0707
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9905 - loss: 0.0648
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 1.0000 - loss: 0.0628
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9931 - loss: 0.0603
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9931 - loss: 0.0591
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 1.0000 - loss: 0.0572
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - accuracy: 1.0000 - loss: 0.0558
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 1.0000 - loss: 0.0546
```

```
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 1.0000 - loss: 0.0534
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 0.0521
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 1.0000 - loss: 0.0509
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 1.0000 - loss: 0.0499
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 0.0488
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 1.0000 - loss: 0.0476
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 1.0000 - loss: 0.0467
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 1.0000 - loss: 0.0458
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 1.0000 - loss: 0.0447
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 1.0000 - loss: 0.0438
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step


Acurácia no treino: 1.0
Acurácia no teste: 0.78
```

Resultado esperado: (pode ser diferente)

```
Acurácia treino = 100.00%
Acurácia teste = 70%
```

Análise dos resultados (5pt)

ToDo: Por que você obteve 100% no treino e apenas 80% no teste no segundo modelo e resultados piores no primeiro modelo?

Isso pode acontecer por causa do overfitting, o segundo modelo aprendeu muito bem os padrões dos dados de treino, com isso o desempenho no treino fica perfeito, quando joga dados novos pra testar ele n consegue manter a mesma precisão. Já o primeiro modelo teve um desempenho pior porque talvez sua arquitetura fosse muito simples e incapaz de capturar padrões mais complexos nos dados.

Modelo 3: Testando com uma rede com três camadas ocultas (15pt)

Definição do modelo (10pt)

In [24]:
```
# Definição do modelo
def modelo_3():
  _model = tf.keras.models.Sequential() # Crie um modelo sequencial com keras.Se
  _model.add(tf.keras.layers.Dense(256,activation = 'relu')) # ToDo: Adicione um
```

```
  _model.add(tf.keras.layers.Dense(64,activation = 'relu')) # ToDo: Adicione uma
  _model.add(tf.keras.layers.Dense(8,activation = 'relu')) # ToDo: Adicione uma
  _model.add(tf.keras.layers.Dense(1,activation = 'sigmoid')) # ToDo: Adicione u

  return _model
```

Crie um modelo com três camadas ocultas e a camada de saída com um neurônio. Você deve seguir a seguinte estrutura:

1. Camada oculta 1 - 256 neurônios e ativação ReLU.
2. Camada oculta 2 - 64 neurônios e ativação ReLU.
3. Camada oculta 3 - 8 neurônios e ativação ReLU.
4. Camada de saída - 1 neurônio e ativação sigmoid.

Agora treine e teste o seu modelo.

In [25]:
```
np.random.seed(1)
tf.random.set_seed(1)

# Criando o modelo
m3 = modelo_3() # ToDo: chame a função que define o modelo
# Treinando o modelo
m3 = treinar_modelo(m3, treino_x, treino_y) # ToDo: Chame a função para treinar

previsoes_treino = np.round(m3.predict(treino_x))
previsoes_test = np.round(m3.predict(teste_x))

## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_score(treino_y.reshape(-1), previsoes_
                                          # **dica** use o model.predict para pre
print(f'Acurácia no teste: {accuracy_score(teste_y.reshape(-1), previsoes_test)}
                                      # **dica** use o model.predict para predizer
```

Model: "sequential_2"

Layer (type)	Output Shape	
dense_4 (Dense)	?	
dense_5 (Dense)	?	
dense_6 (Dense)	?	
dense_7 (Dense)	?	

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 44ms/step - accuracy: 0.5772 - loss: 1.7127
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.6502 - loss: 0.7955
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.6079 - loss: 1.1520
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.6216 - loss: 0.8727
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.6605 - loss: 0.7479
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.6220 - loss: 0.7171
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.6278 - loss: 0.6776
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.6979 - loss: 0.5801
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.7479 - loss: 0.5214
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.7809 - loss: 0.5086
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.7669 - loss: 0.4996
Epoch 12/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 0.7690 - loss: 0.4970
Epoch 13/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.7840 - loss: 0.4904
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.7858 - loss: 0.4851
Epoch 15/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 0.7684 - loss: 0.4781
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.7276 - loss: 0.5168
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.7190 - loss: 0.5493
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.7408 - loss: 0.5009
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.7960 - loss: 0.4330
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8418 - loss: 0.3958
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.8409 - loss: 0.3928
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.8408 - loss: 0.3597
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.8676 - loss: 0.3426
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.8625 - loss: 0.3388
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.8746 - loss: 0.3377
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.8546 - loss: 0.3422
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.9090 - loss: 0.2683
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9036 - loss: 0.2745
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - accuracy: 0.8954 - loss: 0.2903
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.8751 - loss: 0.3106
```

```
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8900 - loss: 0.2874
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.8978 - loss: 0.2928
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.9444 - loss: 0.2381
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9453 - loss: 0.2084
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step - accuracy: 0.9319 - loss: 0.1930
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.8843 - loss: 0.2379
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.8177 - loss: 0.3557
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.8081 - loss: 0.3660
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step - accuracy: 0.9136 - loss: 0.1927
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.9138 - loss: 0.1838
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.9468 - loss: 0.1735
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.9622 - loss: 0.1585
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - accuracy: 0.9680 - loss: 0.1223
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 0.9765 - loss: 0.0980
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9863 - loss: 0.0759
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.9947 - loss: 0.0805
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.9624 - loss: 0.1060
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.9337 - loss: 0.1574
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9084 - loss: 0.2064
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.8745 - loss: 0.2493
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.8966 - loss: 0.2535
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9082 - loss: 0.1930
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.8670 - loss: 0.3262
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.9265 - loss: 0.2200
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9931 - loss: 0.0705
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9911 - loss: 0.0747
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9818 - loss: 0.1121
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.9931 - loss: 0.0580
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.9964 - loss: 0.0527
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step - accuracy: 0.9756 - loss: 0.0812
```

```
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 0.9863 - loss: 0.0673
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 1.0000 - loss: 0.0376
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9863 - loss: 0.0455
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9863 - loss: 0.0417
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 0.0289
Epoch 66/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9931 - loss: 0.0355
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - accuracy: 0.9931 - loss: 0.0312
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 1.0000 - loss: 0.0233
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 1.0000 - loss: 0.0286
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 1.0000 - loss: 0.0241
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 1.0000 - loss: 0.0203
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.9931 - loss: 0.0258
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 1.0000 - loss: 0.0183
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 1.0000 - loss: 0.0174
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 1.0000 - loss: 0.0202
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 1.0000 - loss: 0.0145
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 1.0000 - loss: 0.0149
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 0.0141
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 0.0123
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 1.0000 - loss: 0.0126
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 1.0000 - loss: 0.0115
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 1.0000 - loss: 0.0108
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 1.0000 - loss: 0.0109
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 1.0000 - loss: 0.0099
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 0.0092
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 1.0000 - loss: 0.0093
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - accuracy: 1.0000 - loss: 0.0086
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 1.0000 - loss: 0.0082
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 1.0000 - loss: 0.0082
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 1.0000 - loss: 0.0075
```

```
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 1.0000 - loss: 0.0072
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 1.0000 - loss: 0.0071
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 1.0000 - loss: 0.0067
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 1.0000 - loss: 0.0062
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 88ms/step - accuracy: 1.0000 - loss: 0.0062
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 1.0000 - loss: 0.0060
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 1.0000 - loss: 0.0057
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 1.0000 - loss: 0.0055
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 1.0000 - loss: 0.0055
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 91ms/step - accuracy: 1.0000 - loss: 0.0049
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step


Acurácia no treino: 1.0
Acurácia no teste: 0.76
```

Resultado esperado:

```
Acurácia treino = 100.00%
Acurácia teste = 76%
```

Análise dos resultados (5pt)

ToDo: O resultado com três camadas ocultas foi melhor ou pior do que usa somente uma camada? Tente explicar os motivos.

O desempenho com três camadas ocultas foi pior, talvez o problema tenha sido o overfitting, pois com mais camadas e parâmetros, a rede pode ter aprendido padrões específicos de dados de treino em vez de generalizar para novos exmplos. E redes mais profundas exigem um volume maior de dados para treinar adequadamente. Se os dados disponíveis n forem suficientes, a rede pode acabar ajudatando demais ao conjunto de treino e ter um desempenho pior no teste.

Testando uma rede que você desenvolveu (15pt)

Crie uma arquitetura e treine/teste o seu modelo

Definição do modelo (10pt)

In [26]:
```
# Definição do modelo
def meu_modelo():
```

```
  _model = tf.keras.models.Sequential() # Crie um modelo sequencial com keras.Se
  _model.add(tf.keras.layers.Dense(256,activation = 'relu')) # ToDo: Adicione um
  _model.add(tf.keras.layers.Dense(64,activation = 'relu')) # ToDo: Adicione uma
  _model.add(tf.keras.layers.Dense(64,activation = 'relu')) # ToDo: Adicione uma
  _model.add(tf.keras.layers.Dense(64,activation = 'relu')) # ToDo: Adicione uma
  _model.add(tf.keras.layers.Dense(8,activation = 'relu')) # ToDo: Adicione uma
  _model.add(tf.keras.layers.Dense(1,activation = 'sigmoid')) # ToDo: Adicione u

  return _model
```

In [27]:

```
np.random.seed(1)
tf.random.set_seed(1)

# Criando o modelo
m4 = meu_modelo() # ToDo: chame a função que define o modelo

treino_y = treino_y.reshape(-1).astype(int)
teste_y = teste_y.reshape(-1).astype(int)

# Treinando o modelo
m4.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = m4.fit(treino_x, treino_y, epochs=100, batch_size=32, verbose=1) # ToD
pred_treino = (m4.predict(treino_x) > 0.5).astype(int).reshape(-1)
pred_teste = (m4.predict(teste_x) > 0.5).astype(int).reshape(-1)

treino_accuracy = accuracy_score(treino_y, pred_treino)
teste_accuracy = accuracy_score(teste_y, pred_teste)

## Predição da rede
print(f'\n\nAcurácia no treino: {treino_accuracy}') # ToDo: Utilize a função acc
                                                  # **dica** use o model.predict para pre
print(f'Acurácia no teste: {teste_accuracy}') # ToDo: Utilize a função accuracy_
```

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.5100 - loss: 0.9451
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.6005 - loss: 0.6866
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.6050 - loss: 0.7872
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - accuracy: 0.6247 - loss: 0.7327
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.6494 - loss: 0.7778
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.6289 - loss: 0.6663
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.6974 - loss: 0.6319
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.6642 - loss: 0.6195
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.6820 - loss: 0.6080
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.6646 - loss: 0.6206
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.5858 - loss: 0.6382
Epoch 12/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.7048 - loss: 0.5449
Epoch 13/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.7075 - loss: 0.5481
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 78ms/step - accuracy: 0.7223 - loss: 0.5342
Epoch 15/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.7183 - loss: 0.5378
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.6372 - loss: 0.5970
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 77ms/step - accuracy: 0.7038 - loss: 0.5188
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 90ms/step - accuracy: 0.6705 - loss: 0.5395
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 146ms/step - accuracy: 0.7717 - loss: 0.4810
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 118ms/step - accuracy: 0.7311 - loss: 0.4700
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - accuracy: 0.7518 - loss: 0.4292
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 92ms/step - accuracy: 0.7876 - loss: 0.4190
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.8072 - loss: 0.3999
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 75ms/step - accuracy: 0.8259 - loss: 0.3540
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 88ms/step - accuracy: 0.8612 - loss: 0.3311
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 116ms/step - accuracy: 0.8226 - loss: 0.3653
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.8788 - loss: 0.2804
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.8634 - loss: 0.2672
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 98ms/step - accuracy: 0.8952 - loss: 0.2571
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 124ms/step - accuracy: 0.9135 - loss: 0.2330
```

```
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step - accuracy: 0.8734 - loss: 0.2499
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 251ms/step - accuracy: 0.8839 - loss: 0.2658
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 173ms/step - accuracy: 0.7852 - loss: 0.4043
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 197ms/step - accuracy: 0.6688 - loss: 0.7622
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 106ms/step - accuracy: 0.4186 - loss: 0.7261
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 126ms/step - accuracy: 0.7518 - loss: 0.5821
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 96ms/step - accuracy: 0.7076 - loss: 0.5301
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 114ms/step - accuracy: 0.7425 - loss: 0.5424
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.7334 - loss: 0.4971
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.8435 - loss: 0.3869
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.7546 - loss: 0.4560
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.7151 - loss: 0.5121
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.8945 - loss: 0.4335
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.8737 - loss: 0.3262
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.7978 - loss: 0.3928
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.7107 - loss: 0.4610
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.9142 - loss: 0.2775
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - accuracy: 0.9063 - loss: 0.2771
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.7463 - loss: 0.4230
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.8136 - loss: 0.3804
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.6998 - loss: 0.4289
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - accuracy: 0.8902 - loss: 0.2855
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.7918 - loss: 0.3281
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step - accuracy: 0.9445 - loss: 0.1929
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 78ms/step - accuracy: 0.9315 - loss: 0.1947
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 74ms/step - accuracy: 0.8892 - loss: 0.2614
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.8399 - loss: 0.3447
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step - accuracy: 0.9060 - loss: 0.2395
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.9293 - loss: 0.1801
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.8823 - loss: 0.2553
```

```
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 87ms/step - accuracy: 0.7900 - loss: 0.4056
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 0.8654 - loss: 0.2915
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9280 - loss: 0.1992
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.9389 - loss: 0.1529
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9450 - loss: 0.1644
Epoch 66/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.9071 - loss: 0.2050
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.8817 - loss: 0.2480
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.7505 - loss: 0.4386
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9100 - loss: 0.2177
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.9699 - loss: 0.1260
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.9478 - loss: 0.1322
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9745 - loss: 0.0712
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step - accuracy: 0.9840 - loss: 0.0670
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.9623 - loss: 0.0795
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9057 - loss: 0.1740
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.8804 - loss: 0.2568
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.8250 - loss: 0.3469
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.8421 - loss: 0.2741
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.9753 - loss: 0.1092
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.9743 - loss: 0.0873
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9468 - loss: 0.1229
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.9477 - loss: 0.1125
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - accuracy: 0.9755 - loss: 0.0515
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 102ms/step - accuracy: 0.9892 - loss: 0.0353
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.9866 - loss: 0.0342
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.9797 - loss: 0.0351
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.9974 - loss: 0.0236
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 1.0000 - loss: 0.0120
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 1.0000 - loss: 0.0141
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.9853 - loss: 0.0329
```

```
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.9368 - loss: 0.1525
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step - accuracy: 0.9672 - loss: 0.0997
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.9851 - loss: 0.0489
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9874 - loss: 0.0387
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - accuracy: 0.9692 - loss: 0.0877
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.9466 - loss: 0.1563
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9381 - loss: 0.1331
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9931 - loss: 0.0283
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 1.0000 - loss: 0.0209
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 1.0000 - loss: 0.0240
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step


Acurácia no treino: 1.0
Acurácia no teste: 0.8
```

Análise dos resultados (5pt)

ToDo: O que você pode falar do seu modelo? Como ele se saiu em relação aos outros três modelos?

ele foi preciso demais e deu overfiting

Variando alguns hiperparâmetros (35pt)

Usando o framework do tensorflow/keras, altere os hiperparâmetros e veja o impacto (gere pelo menos dois novos modelos):

- *Learning Rate*.
- Algoritmo de otimização (SGD com momento, ADAM, ADADELTA, RMSPROP).
- inicialização dos pesos: inicialiação aleatória vs uniforme.
- Funções de ativação : troque a sigmoid por (ReLU, GELU, Leaky RELU).

Cria a sua própria função de treinamento (15pt)

Você deve criar uma nova função para treinamento. Essa nova função, deve receber os parâmetros que você irá alterar, como por exemplo, *Learning Rate* e otimizador.

In [28]:
```
# Função para treinamento do modelo
def treinar_modelo(modelo, treino_x, treino_y, epochs=100):  # ToDo: Adicione os
    # Setando a seed
    np.random.seed(1)
```

```
    tf.random.set_seed(1)

    # Compilando o modelo
    modelo.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur

    # Imprimindo a arquitetura da rede proposta
    modelo.summary()

    # Treinando o modelo
    history = modelo.fit(treino_x, treino_y, epochs=epochs, batch_size=32, verbo

    return modelo, history
```

Desenvolva os seus modelos aqui e os teste nos dados de teste (15pt)

In [29]:

```
### Início do código para o Modelo 1 ###
modelo_1 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, activation='relu', input_shape=(treino_x.shape[1]
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
### Fim do código para o Modelo 1 ###

### Início do código para o Modelo 2 ###
modelo_2 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(treino_x.shape[1]
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
### Fim do código para o Modelo 2 ###

# Treinando o Modelo 1
modelo_treinado_1, history_1 = treinar_modelo(modelo_1, treino_x, treino_y, epoc

# Treinando o Modelo 2
modelo_treinado_2, history_2 = treinar_modelo(modelo_2, treino_x, treino_y, epoc
# Predições para o Modelo 1
pred_treino_1 = (modelo_treinado_1.predict(treino_x) > 0.5).astype(int).reshape(
pred_teste_1 = (modelo_treinado_1.predict(teste_x) > 0.5).astype(int).reshape(-1

# Acurácias para o Modelo 1
treino_accuracy_1 = accuracy_score(treino_y, pred_treino_1)
teste_accuracy_1 = accuracy_score(teste_y, pred_teste_1)

# Predições para o Modelo 2
pred_treino_2 = (modelo_treinado_2.predict(treino_x) > 0.5).astype(int).reshape(
pred_teste_2 = (modelo_treinado_2.predict(teste_x) > 0.5).astype(int).reshape(-1

# Acurácias para o Modelo 2
treino_accuracy_2 = accuracy_score(treino_y, pred_treino_2)
teste_accuracy_2 = accuracy_score(teste_y, pred_teste_2)

# Imprimindo as Acurácias
print(f'\n\nAcurácia no treino do Modelo 1: {treino_accuracy_1}')
print(f'Acurácia no teste do Modelo 1: {teste_accuracy_1}\n')
```

```
print(f'Acurácia no treino do Modelo 2: {treino_accuracy_2}')
print(f'Acurácia no teste do Modelo 2: {teste_accuracy_2}')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
equential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_4"

Layer (type)	Output Shape	
dense_14 (Dense)	(None, 256)	
dense_15 (Dense)	(None, 64)	
dense_16 (Dense)	(None, 1)	

Total params: 3,162,497 (12.06 MB)

Trainable params: 3,162,497 (12.06 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.6021 - loss: 3.1949
Epoch 2/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - accuracy: 0.5984 - loss: 1.2009
Epoch 3/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.6362 - loss: 0.6513
Epoch 4/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.6932 - loss: 0.6227
Epoch 5/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 0.6208 - loss: 0.8805
Epoch 6/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.6689 - loss: 1.0840
Epoch 7/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.6208 - loss: 0.8457
Epoch 8/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 90ms/step - accuracy: 0.6510 - loss: 0.6477
Epoch 9/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.7740 - loss: 0.5034
Epoch 10/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.7928 - loss: 0.4708
Epoch 11/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.7712 - loss: 0.4604
Epoch 12/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.7507 - loss: 0.5011
Epoch 13/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.7768 - loss: 0.5002
Epoch 14/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.7468 - loss: 0.5008
Epoch 15/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.7602 - loss: 0.5110
Epoch 16/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.7348 - loss: 0.5319
Epoch 17/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.7336 - loss: 0.5489
Epoch 18/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.7487 - loss: 0.4887
Epoch 19/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.7900 - loss: 0.4417
Epoch 20/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8121 - loss: 0.3900
Epoch 21/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.8510 - loss: 0.3757
Epoch 22/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.8884 - loss: 0.3320
Epoch 23/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 0.9360 - loss: 0.2993
Epoch 24/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9262 - loss: 0.2767
Epoch 25/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.9425 - loss: 0.2586
Epoch 26/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.9263 - loss: 0.2496
Epoch 27/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.9324 - loss: 0.2378
Epoch 28/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.9167 - loss: 0.2410
Epoch 29/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8938 - loss: 0.2458
Epoch 30/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.8826 - loss: 0.2597
```

```
Epoch 31/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.8554 - loss: 0.2913
Epoch 32/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 88ms/step - accuracy: 0.8182 - loss: 0.3567
Epoch 33/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.7915 - loss: 0.4401
Epoch 34/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 76ms/step - accuracy: 0.8261 - loss: 0.4590
Epoch 35/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.8436 - loss: 0.4424
Epoch 36/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.7399 - loss: 0.6775
Epoch 37/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 123ms/step - accuracy: 0.7405 - loss: 0.7435
Epoch 38/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 117ms/step - accuracy: 0.6893 - loss: 0.6062
Epoch 39/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 77ms/step - accuracy: 0.8576 - loss: 0.3410
Epoch 40/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.8789 - loss: 0.3032
Epoch 41/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9022 - loss: 0.2588
Epoch 42/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - accuracy: 0.9446 - loss: 0.1876
Epoch 43/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.9536 - loss: 0.1585
Epoch 44/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.9723 - loss: 0.1305
Epoch 45/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 0.9905 - loss: 0.1183
Epoch 46/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.9879 - loss: 0.1145
Epoch 47/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9869 - loss: 0.1117
Epoch 48/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.9869 - loss: 0.1093
Epoch 49/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9643 - loss: 0.1174
Epoch 50/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9686 - loss: 0.1233
Model: "sequential_5"
```

Layer (type)	Output Shape	
dense_17 (Dense)	(None, 128)	
dropout (Dropout)	(None, 128)	
dense_18 (Dense)	(None, 64)	
dense_19 (Dense)	(None, 1)	

```
Total params: 1,581,313 (6.03 MB)
Trainable params: 1,581,313 (6.03 MB)
Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.5062 - loss: 1.3813
Epoch 2/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.5930 - loss: 1.1909
Epoch 3/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.6196 - loss: 1.0365
Epoch 4/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.5606 - loss: 1.2320
Epoch 5/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - accuracy: 0.5599 - loss: 0.9576
Epoch 6/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.6048 - loss: 0.7569
Epoch 7/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.6504 - loss: 0.6852
Epoch 8/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.6351 - loss: 0.7134
Epoch 9/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.6746 - loss: 0.7076
Epoch 10/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.6016 - loss: 0.7013
Epoch 11/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.6124 - loss: 0.6813
Epoch 12/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.6681 - loss: 0.6473
Epoch 13/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.6494 - loss: 0.6671
Epoch 14/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.5971 - loss: 0.6773
Epoch 15/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.6815 - loss: 0.6178
Epoch 16/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.6504 - loss: 0.5589
Epoch 17/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.6841 - loss: 0.5901
Epoch 18/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.6911 - loss: 0.5698
Epoch 19/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.7438 - loss: 0.5633
Epoch 20/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.6643 - loss: 0.5819
Epoch 21/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.6791 - loss: 0.5477
Epoch 22/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.6608 - loss: 0.5709
Epoch 23/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.6943 - loss: 0.5473
Epoch 24/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.6956 - loss: 0.5350
Epoch 25/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.7133 - loss: 0.5509
Epoch 26/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7063 - loss: 0.5613
Epoch 27/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.7227 - loss: 0.5461
Epoch 28/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.6495 - loss: 0.6201
Epoch 29/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.7346 - loss: 0.5326
Epoch 30/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.6861 - loss: 0.5169
```

```
Epoch 31/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.7630 - loss: 0.4703
Epoch 32/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.7223 - loss: 0.5033
Epoch 33/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.7521 - loss: 0.4896
Epoch 34/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.7663 - loss: 0.4943
Epoch 35/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7887 - loss: 0.4597
Epoch 36/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7665 - loss: 0.4438
Epoch 37/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7837 - loss: 0.4469
Epoch 38/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.7933 - loss: 0.4169
Epoch 39/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.8002 - loss: 0.4239
Epoch 40/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.7981 - loss: 0.4015
Epoch 41/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.8104 - loss: 0.3995
Epoch 42/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.7824 - loss: 0.4464
Epoch 43/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.8067 - loss: 0.3726
Epoch 44/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.8467 - loss: 0.3954
Epoch 45/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.7703 - loss: 0.4126
Epoch 46/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.8685 - loss: 0.3309
Epoch 47/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.8400 - loss: 0.3595
Epoch 48/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.8148 - loss: 0.3841
Epoch 49/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.7783 - loss: 0.4092
Epoch 50/50
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.7367 - loss: 0.4210
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step


Acurácia no treino do Modelo 1: 0.9425837320574163
Acurácia no teste do Modelo 1: 0.78

Acurácia no treino do Modelo 2: 0.8277511961722488
Acurácia no teste do Modelo 2: 0.5
```

Analisando as redes treinadas (5pt)

ToDo: Qual combinação rendeu o melhor resultado? Tente explicar o por que.

A combinação do modelo 1 usando 256 foi melhor, creio que seja devido a quantidade de parâmetros usados, aumentando sua precisão.

Analisando outras métricas (10pt)

Nem sempre somente a acurácia é uma boa análise. Outras métricas podem ser úteis, como precisão, revocação e F1-Score. Para isso, considere os quatro modelos criados e os outros que você desenvolveu e avalie as métricas precisão, revocação e F1-Score.

In [30]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

Desenvolva o código para calcular as métricas (5pt)

Após a importação do pacote, avalie cada uma das métricas para **todos** os modelos somente nos dados de teste.

In [32]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Cálculo das métricas para o Modelo 1
precision_1 = precision_score(teste_y, pred_teste_1)
recall_1 = recall_score(teste_y, pred_teste_1)
f1_1 = f1_score(teste_y, pred_teste_1)

print(f'Precisão do Modelo 1: {precision_1}')
print(f'Recall do Modelo 1: {recall_1}')
print(f'F1-Score do Modelo 1: {f1_1}')

# Cálculo das métricas para o Modelo 2
precision_2 = precision_score(teste_y, pred_teste_2)
recall_2 = recall_score(teste_y, pred_teste_2)
f1_2 = f1_score(teste_y, pred_teste_2)

print(f'Precisão do Modelo 2: {precision_2}')
print(f'Recall do Modelo 2: {recall_2}')
print(f'F1-Score do Modelo 2: {f1_2}')
```

```
Precisão do Modelo 1: 0.8235294117647058
Recall do Modelo 1: 0.8484848484848485
F1-Score do Modelo 1: 0.835820895522388
Precisão do Modelo 2: 0.8333333333333334
Recall do Modelo 2: 0.30303030303030304
F1-Score do Modelo 2: 0.4444444444444444
```

Analisando o treinamento dos modelos

ToDo: O que você pode falar sobre os modelos treinados e as métricas avaliadas? (5pt)

> Para uma análise mais completa, adicionamos métricas como precisão, recall e F1-score, que ajudam a entender melhor o desempenho dos modelos. A precisão nos mostra quantas previsões positivas estavam corretas, o recall indica quantos dos casos realmente positivos foram identificados, e o F1-score equilibra essas duas métricas. Isso é importante porque um modelo pode ter alta acurácia, mas ainda cometer muitos erros ao classificar

casos específicos. Comparando esses valores entre os modelos, podemos entender melhor qual deles é mais eficiente e se há necessidade de ajustes, como modificar a estrutura da rede ou otimizar hiperparâmetros para melhorar os resultados.