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*.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import h5py
4 import scipy
5
6 from sklearn.metrics import accuracy_score
7
8 from tensorflow import keras
9 import tensorflow as tf
```

Configurando os *plots* de gráficos

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

```
1 %matplotlib inline
2 plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of plots
3 plt.rcParams['image.interpolation'] = 'nearest'
4 plt.rcParams['image.cmap'] = 'gray'
5
6 %load_ext autoreload
7 %autoreload 2
8
9 np.random.seed(1)
```

Configurando o Google Colab.

Configurando o Google Colab para acessar os nossos dados.

```
1 # Você vai precisar fazer o upload dos arquivos no seu drive (faer na pasta raiz) e mo
2 # não se esqueça de ajustar o path para o seu drive
3 from google.colab import drive
4 drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

Carregando e prépprocessamento dos dados

Carregando e preprocessamento dos dados

```
# Função para ler os dados (gato/não-gato)
def load_dataset():
    def _load_data():
      train_dataset = h5py.File('/content/drive/MyDrive/Praticas redes neurais/Lab2/tr
      train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train set fe
      train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train set la

      test_dataset = h5py.File('/content/drive/MyDrive/Praticas redes neurais/Lab2/tes
      test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set featur
      test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set labels

      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_orig, cla

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

      # Normalize os dados (colocar no intervalo [0.0, 1.0])
      _treino_x = _treino_x_vet/255. # ToDo: normalize os dados de treinamento aqui
      _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
```

Carregando os dados

```
# Lendo os dados (gato/não-gato)
treino_x, treino_y, teste_x, teste_y, classes = load_dataset()
```

Treinamento do modelo (100pt)

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

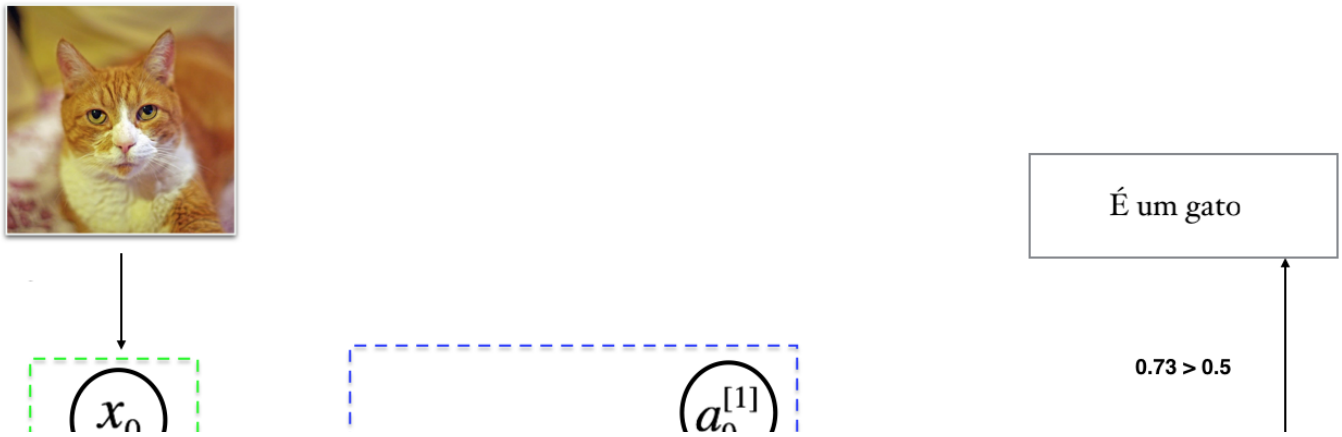
Função para treinar um modelo

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.

```
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).



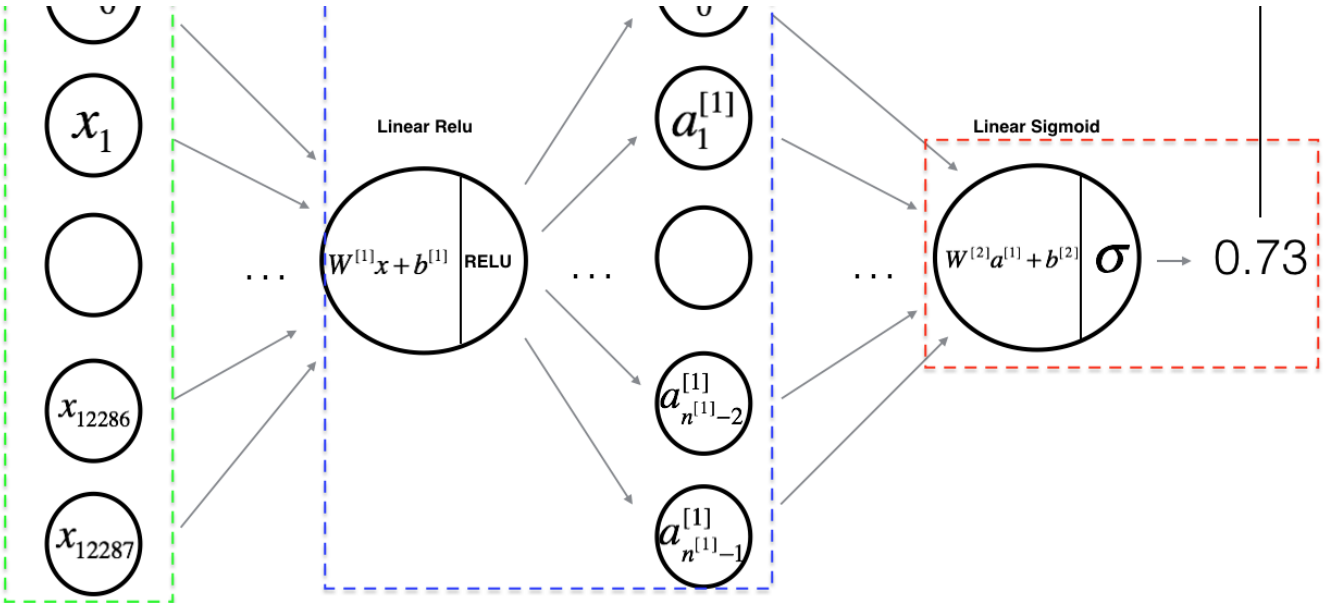


Figura 1: Rede neural com 2 camadas.
Resumo do modelo: ***ENTRADA -> LINEAR -> RELU -> LINEAR -> SIGMOID -> SAIDA***.

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.

```
# Definição do modelo
def modelo_1():
  _model = keras.Sequential() # Crie um modelo sequencial com keras.Sequential
  _model.add(keras.Input(treino_x[0].shape))
  _model.add(keras.layers.Dense(8, activation="relu")) # ToDo: Adicione uma camada den
  _model.add(keras.layers.Dense(1, activation="sigmoid")) # ToDo: Adicione uma camada
  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.

```
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 o mode

## Predição da rede
m1_train_predictions = (m1.predict(treino_x) > 0.5).astype(int).reshape(-1)
train_accuracy = accuracy_score(treino_y.reshape(-1), m1_train_predictions)
print(f'\n\nAcurácia no treino: {train_accuracy}') # ToDo: Utilize a função accuracy_s
                                        # **dica** use o model.predict para predizer

m1_test_predictions = (m1.predict(teste_x) > 0.5).astype(int).reshape(-1)
test_accuracy = accuracy_score(teste_y.reshape(-1), m1_test_predictions)
print(f'Acurácia no teste: {test_accuracy}') # ToDo: Utilize a função accuracy_score d
                                    # **dica** use o model.predict para predizer os da
```

Model: "sequential"

Layer (type)	Output Shape	Param
dense (Dense)	(None, 8)	98,31
dense_1 (Dense)	(None, 1)	

Total params: 98,321 (384.07 KB)
Trainable params: 98,321 (384.07 KB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 7s 168ms/step - accuracy: 0.5154 - loss: 0.7893
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6601 - loss: 0.6373
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6716 - loss: 0.6017
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7224 - loss: 0.5870
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7068 - loss: 0.5665
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7037 - loss: 0.5535
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7434 - loss: 0.5307
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7933 - loss: 0.5025
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8055 - loss: 0.4796
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8130 - loss: 0.4632
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8338 - loss: 0.4506
Epoch 12/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8221 - loss: 0.4409
Epoch 13/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8334 - loss: 0.4331
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8272 - loss: 0.4263
Epoch 15/100
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8164 - loss: 0.4206
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8257 - loss: 0.4168
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8406 - loss: 0.4158
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8378 - loss: 0.4175
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8341 - loss: 0.4196
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8209 - loss: 0.4178
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8356 - loss: 0.4086
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8418 - loss: 0.3895
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8632 - loss: 0.3643
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9070 - loss: 0.3396
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9213 - loss: 0.3195
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9235 - loss: 0.3048
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9143 - loss: 0.2953
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9211 - loss: 0.2901
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9213 - loss: 0.2873
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9177 - loss: 0.2852
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9109 - loss: 0.2824
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9141 - loss: 0.2783
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9105 - loss: 0.2751
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8993 - loss: 0.2771
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9036 - loss: 0.2908
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8653 - loss: 0.3209
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8366 - loss: 0.3651
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8304 - loss: 0.4017
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8469 - loss: 0.3899
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8525 - loss: 0.3335
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9188 - loss: 0.2735
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9414 - loss: 0.2359
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9414 - loss: 0.2599
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9432 - loss: 0.2260
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9345 - loss: 0.2196
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9472 - loss: 0.2066
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9504 - loss: 0.1975
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9559 - loss: 0.1983
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9445 - loss: 0.2060
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9410 - loss: 0.2126
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9479 - loss: 0.2124
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9639 - loss: 0.2041
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9597 - loss: 0.1912
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9642 - loss: 0.1778
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9642 - loss: 0.1674
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9729 - loss: 0.1614
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9703 - loss: 0.1583
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9771 - loss: 0.1552
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9797 - loss: 0.1506
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9729 - loss: 0.1452
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9729 - loss: 0.1420
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9797 - loss: 0.1436
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9729 - loss: 0.1500
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9703 - loss: 0.1593
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9658 - loss: 0.1685
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9587 - loss: 0.1749
Epoch 66/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9587 - loss: 0.1760
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9544 - loss: 0.1706
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9692 - loss: 0.1589
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9718 - loss: 0.1430
```

```
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9779 - loss: 0.1305
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9595 - loss: 0.1359
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9285 - loss: 0.1591
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9074 - loss: 0.1751
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9318 - loss: 0.1637
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9602 - loss: 0.1334
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9928 - loss: 0.1154
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9833 - loss: 0.1354
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9408 - loss: 0.1749
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9162 - loss: 0.2023
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9234 - loss: 0.2026
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9508 - loss: 0.1766
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9669 - loss: 0.1347
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9703 - loss: 0.1122
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9663 - loss: 0.1255
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9660 - loss: 0.1094
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0812
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0798
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9931 - loss: 0.0797
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0746
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0724
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0704
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0685
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0673
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0659
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9931 - loss: 0.0644
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0631
```

```
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0619
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0607
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9931 - loss: 0.0596
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9931 - loss: 0.0585
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step


Acurácia no treino: 0.9952153110047847
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
Acurácia no teste: 0.72
```

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)

```
# Definição do modelo
def modelo_2():
  _model = keras.Sequential()
  _model.add(keras.Input(treino_x[0].shape))
  _model.add(keras.layers.Dense(256, activation = "relu"))
  _model.add(keras.layers.Dense(1, activation = "sigmoid"))
  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.

```
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 o mode

## Predição da rede
m2_train_predictions = (m2.predict(treino_x) > 0.5).astype(int).reshape(-1)
train_accuracy = accuracy_score(treino_y.reshape(-1), m2_train_predictions)
print(f'\n\nAcurácia no treino: {train_accuracy}') # ToDo: Utilize a função accuracy_s
                                        # **dica** use o model.predict para predizer

m2_test_predictions = (m2.predict(teste_x) > 0.5).astype(int).reshape(-1)
test_accuracy = accuracy_score(teste_y.reshape(-1), m2_test_predictions)
print(f'Acurácia no teste: {test_accuracy}') # ToDo: Utilize a função accuracy_score d
                                    # **dica** use o model.predict para predizer os da
```

Model: "sequential_1"

Layer (type)	Output Shape	Param
dense_2 (Dense)	(None, 256)	3,145,98
dense_3 (Dense)	(None, 1)	25

Total params: 3,146,241 (12.00 MB)
Trainable params: 3,146,241 (12.00 MB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 86ms/step - accuracy: 0.4773 - loss: 4.7217
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3675 - loss: 1.8977
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6920
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6913
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6905
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6897
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6724 - loss: 0.6895
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6854 - loss: 0.6859
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6878
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6678 - loss: 0.6872
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6678 - loss: 0.6866
Epoch 12/100
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6859
Epoch 13/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6851
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6844
Epoch 15/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6836
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6828
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6819
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6678 - loss: 0.6811
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6678 - loss: 0.6803
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6678 - loss: 0.6793
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6854 - loss: 0.6764
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6769 - loss: 0.6678
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7010 - loss: 0.6668
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7121 - loss: 0.6569
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7350 - loss: 0.6502
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7054 - loss: 0.6488
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7115 - loss: 0.6624
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7526 - loss: 0.6342
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7515 - loss: 0.6252
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7497 - loss: 0.6191
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7732 - loss: 0.6115
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7805 - loss: 0.6072
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7964 - loss: 0.5993
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7816 - loss: 0.5998
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7600 - loss: 0.5969
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7737 - loss: 0.5918
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7999 - loss: 0.5817
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7935 - loss: 0.5786
Epoch 39/100
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8003 - loss: 0.5727
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7946 - loss: 0.5852
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7960 - loss: 0.5744
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8118 - loss: 0.5690
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7568 - loss: 0.5714
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7950 - loss: 0.5592
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8028 - loss: 0.5577
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8009 - loss: 0.5563
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8128 - loss: 0.5494
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8185 - loss: 0.5551
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7969 - loss: 0.5498
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8159 - loss: 0.5420
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8221 - loss: 0.5398
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8221 - loss: 0.5379
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8221 - loss: 0.5387
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8247 - loss: 0.5315
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.5309
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8163 - loss: 0.5310
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8247 - loss: 0.5254
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8297 - loss: 0.5229
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8270 - loss: 0.5220
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8202 - loss: 0.5219
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8247 - loss: 0.5199
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8297 - loss: 0.5161
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8244 - loss: 0.5155
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.5197
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.5201
Epoch 66/100
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8221 - loss: 0.5133
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8172 - loss: 0.5313
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8198 - loss: 0.5112
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.5107
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8055 - loss: 0.5215
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8285 - loss: 0.5068
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8221 - loss: 0.5170
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8203 - loss: 0.5052
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.5108
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8247 - loss: 0.5014
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8270 - loss: 0.5033
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.5005
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8055 - loss: 0.5084
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.5006
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8163 - loss: 0.5072
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.5000
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7947 - loss: 0.5070
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.5048
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8247 - loss: 0.4952
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.4952
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8221 - loss: 0.4908
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.4948
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.4879
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.4899
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.4867
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8247 - loss: 0.4872
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8297 - loss: 0.4863
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.4845
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.4845
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.4964
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8244 - loss: 0.4842
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8270 - loss: 0.4901
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8278 - loss: 0.4808
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.4819
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8297 - loss: 0.4793
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8247 - loss: 0.4780
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step


Acurácia no treino: 0.8133971291866029
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
Acurácia no teste: 0.58
```

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?

```
O segundo modelo possui uma maior quantidade de neurônios e, consequentemente, uma maior capacidad
```

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

Definição do modelo (10pt)

```
# Definição do modelo
def modelo_3():
  _model = keras.Sequential()
  _model.add(keras.Input(treino_x[0].shape))
  _model.add(keras.layers.Dense(256, activation = "relu"))
  _model.add(keras.layers.Dense(64, activation = "relu"))
  _model.add(keras.layers.Dense(8, activation = "relu"))
  _model.add(keras.layers.Dense(1, activation = "sigmoid"))
  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.

```
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 o mode

## Predição da rede
m3_train_predictions = (m3.predict(treino_x) > 0.5).astype(int).reshape(-1)
train_accuracy = accuracy_score(treino_y.reshape(-1), m3_train_predictions)
print(f'\n\nAcurácia no treino: {train_accuracy}') # ToDo: Utilize a função accuracy_s
                                        # **dica** use o model.predict para predizer

m3_test_predictions = (m3.predict(teste_x) > 0.5).astype(int).reshape(-1)
test_accuracy = accuracy_score(teste_y.reshape(-1), m3_test_predictions)
print(f'Acurácia no teste: {test_accuracy}') # ToDo: Utilize a função accuracy_score d
                                    # **dica** use o model.predict para predizer os da
```

Model: "sequential_2"

Layer (type)	Output Shape	Param
dense_4 (Dense)	(None, 256)	3,145,98

dense_5 (Dense)	(None, 64)	16,44
dense_6 (Dense)	(None, 8)	52
dense_7 (Dense)	(None, 1)	

Total params: 3,162,961 (12.07 MB)
Trainable params: 3,162,961 (12.07 MB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 0.5425 - loss: 0.9362
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6603 - loss: 0.6584
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6609 - loss: 0.6572
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6750 - loss: 0.6344
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6585 - loss: 0.6190
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7007 - loss: 0.6168
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6951 - loss: 0.5972
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6999 - loss: 0.5854
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7106 - loss: 0.5793
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7174 - loss: 0.5674
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7300 - loss: 0.5531
Epoch 12/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7298 - loss: 0.5414
Epoch 13/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7413 - loss: 0.5309
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7536 - loss: 0.5209
Epoch 15/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7554 - loss: 0.5103
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7706 - loss: 0.4985
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7596 - loss: 0.4885
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7800 - loss: 0.4781
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7776 - loss: 0.4687
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7825 - loss: 0.4578
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7981 - loss: 0.4479
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8151 - loss: 0.4414
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8281 - loss: 0.4373
```

```
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8324 - loss: 0.4344
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8333 - loss: 0.4359
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8096 - loss: 0.4492
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7874 - loss: 0.4715
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7843 - loss: 0.4625
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7833 - loss: 0.4139
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8527 - loss: 0.3542
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8901 - loss: 0.3094
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9092 - loss: 0.2896
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9054 - loss: 0.2798
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9062 - loss: 0.2757
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8928 - loss: 0.2726
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8867 - loss: 0.2655
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9053 - loss: 0.2511
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9024 - loss: 0.2341
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9065 - loss: 0.2217
Epoch 40/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9107 - loss: 0.2261
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8794 - loss: 0.2530
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8436 - loss: 0.3056
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8142 - loss: 0.4013
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7658 - loss: 0.5285
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7791 - loss: 0.5226
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8301 - loss: 0.3378
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9027 - loss: 0.2218
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9464 - loss: 0.1773
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9658 - loss: 0.1506
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9589 - loss: 0.1561
```

```
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9606 - loss: 0.1394
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9719 - loss: 0.1234
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9719 - loss: 0.1141
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9719 - loss: 0.1082
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9788 - loss: 0.1025
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9895 - loss: 0.0975
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9895 - loss: 0.0924
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9895 - loss: 0.0852
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9895 - loss: 0.0782
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9931 - loss: 0.0718
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9931 - loss: 0.0655
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0597
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9964 - loss: 0.0549
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0490
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0463
Epoch 66/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9931 - loss: 0.0433
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9931 - loss: 0.0433
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9931 - loss: 0.0460
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9837 - loss: 0.0598
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9764 - loss: 0.0871
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9524 - loss: 0.1119
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9591 - loss: 0.0904
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9498 - loss: 0.1406
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9436 - loss: 0.1546
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8956 - loss: 0.1966
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9065 - loss: 0.2383
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8320 - loss: 0.3776
```

```
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8349 - loss: 0.4678
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7753 - loss: 0.7706
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7692 - loss: 0.6606
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8419 - loss: 0.4931
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9167 - loss: 0.2084
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9412 - loss: 0.1659
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9322 - loss: 0.1468
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9882 - loss: 0.0907
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9931 - loss: 0.0666
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9931 - loss: 0.0754
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0447
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0463
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0382
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0375
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0351
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0326
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0309
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0289
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0278
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0260
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0248
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0234
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0214
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step


Acurácia no treino: 1.0
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 278ms/step
Acurácia no teste: 0.74
```

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 resultado no treino foi melhor, uma vez que o modelo conseguiu capturar bem as características d
```

Testando uma rede que você desenvolveu (15pt)

Crie uma arquitetura e treine/teste o seu modelo

Definição do modelo (10pt)

```
# Definição do modelo
def meu_modelo():
  _model = keras.Sequential()
  _model.add(keras.layers.Dense(512, activation="relu"))
  _model.add(keras.layers.Dense(128, activation="relu"))
  _model.add(keras.layers.BatchNormalization())
  _model.add(keras.layers.Dense(32, activation="relu"))
  _model.add(keras.layers.Dense(1, activation="sigmoid"))
  return _model
```

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

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

# Treinando o modelo
m4 = treinar_modelo(m4, treino_x, treino_y) # ToDo: Chame a função para treinar o mode
```

```
 9
10 ## Predição da rede
11 m4_train_predictions = (m4.predict(treino_x) > 0.5).astype(int).reshape(-1)
12 train_accuracy = accuracy_score(treino_y.reshape(-1), m4_train_predictions)
13 print(f'\n\nAcurácia no treino: {train_accuracy}') # ToDo: Utilize a função accuracy_s
14                                         # **dica** use o model.predict para predizer
15
16 m4_test_predictions = (m4.predict(teste_x) > 0.5).astype(int).reshape(-1)
17 test_accuracy = accuracy_score(teste_y.reshape(-1), m4_test_predictions)
18 print(f'Acurácia no teste: {test_accuracy}') # ToDo: Utilize a função accuracy_score d
19                                       # **dica** use o model.predict para predizer os da
```

Model: "sequential_3"

Layer (type)	Output Shape	Param
dense_8 (Dense)	?	0 (unbuilt
dense_9 (Dense)	?	0 (unbuilt
batch_normalization (BatchNormalization)	?	0 (unbuilt
dense_10 (Dense)	?	0 (unbuilt
dense_11 (Dense)	?	0 (unbuilt

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 168ms/step - accuracy: 0.5040 - loss: 0.7660
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7857 - loss: 0.5396
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8305 - loss: 0.4856
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8401 - loss: 0.4366
Epoch 5/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8720 - loss: 0.3940
Epoch 6/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9048 - loss: 0.3425
Epoch 7/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9253 - loss: 0.2956
Epoch 8/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9498 - loss: 0.2623
Epoch 9/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9667 - loss: 0.2626
Epoch 10/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9705 - loss: 0.2281
Epoch 11/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9565 - loss: 0.2023
Epoch 12/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9794 - loss: 0.1766
Epoch 13/100
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9761 - loss: 0.1520
Epoch 14/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9856 - loss: 0.1264
Epoch 15/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9856 - loss: 0.1010
Epoch 16/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0771
Epoch 17/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0516
Epoch 18/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0385
Epoch 19/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0302
Epoch 20/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0225
Epoch 21/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0170
Epoch 22/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0133
Epoch 23/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0112
Epoch 24/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0096
Epoch 25/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0083
Epoch 26/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0073
Epoch 27/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0065
Epoch 28/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0058
Epoch 29/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0052
Epoch 30/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0047
Epoch 31/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0043
Epoch 32/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0039
Epoch 33/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0037
Epoch 34/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0034
Epoch 35/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0032
Epoch 36/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0030
Epoch 37/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0028
Epoch 38/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0027
Epoch 39/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0025
Epoch 40/100
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0024
Epoch 41/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0022
Epoch 42/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 43/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0020
Epoch 44/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0019
Epoch 45/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0018
Epoch 46/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0018
Epoch 47/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0017
Epoch 48/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0016
Epoch 49/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0015
Epoch 50/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0015
Epoch 51/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0014
Epoch 52/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0014
Epoch 53/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0013
Epoch 54/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0013
Epoch 55/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0012
Epoch 56/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0012
Epoch 57/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 58/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 59/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0010
Epoch 60/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0010
Epoch 61/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 9.7854e-04
Epoch 62/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 9.4841e-04
Epoch 63/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 9.1467e-04
Epoch 64/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 8.8913e-04
Epoch 65/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 8.5898e-04
Epoch 66/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 8.3440e-04
Epoch 67/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 8.0667e-04
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 8.0687e-04
Epoch 68/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 7.8422e-04
Epoch 69/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 7.5929e-04
Epoch 70/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 7.3784e-04
Epoch 71/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 7.1540e-04
Epoch 72/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 6.9701e-04
Epoch 73/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 6.7585e-04
Epoch 74/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 6.5853e-04
Epoch 75/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 6.3927e-04
Epoch 76/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 6.2303e-04
Epoch 77/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 6.0552e-04
Epoch 78/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.9088e-04
Epoch 79/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.7468e-04
Epoch 80/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.6127e-04
Epoch 81/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.4596e-04
Epoch 82/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.3256e-04
Epoch 83/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.1939e-04
Epoch 84/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.0741e-04
Epoch 85/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.9426e-04
Epoch 86/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.8305e-04
Epoch 87/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.7078e-04
Epoch 88/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.6033e-04
Epoch 89/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.4903e-04
Epoch 90/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.3909e-04
Epoch 91/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.2869e-04
Epoch 92/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.1943e-04
Epoch 93/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.0947e-04
Epoch 94/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 4.0081e-04
```

```
Epoch 95/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 3.9173e-04
Epoch 96/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 3.8331e-04
Epoch 97/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 3.7497e-04
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 3.6708e-04
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 3.5917e-04
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 3.5174e-04
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step


Acurácia no treino: 1.0
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 384ms/step
Acurácia no teste: 0.76
```

Análise dos resultados (5pt)

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

O modelo desenvolvido apresenta uma acurácia semelhante no treino, 100%, porém uma ligeira melhora

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.

```
def treinar_modelo(modelo, treino_x, treino_y, epochs=100, learning_rate = 0.001, otim
    # Setando a seed
    np.random.seed(1)

    # Compilando o modelo
    modelo.compile(optimizer= otimizador(learning_rate=learning_rate),
                  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, verbose = 0)
    return modelo

```

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

```
from copy import deepcopy
### Início do código para o Modelo 1 ###
modelo1 = keras.Sequential()
modelo1.add(keras.Input(treino_x[0].shape))
modelo1.add(keras.layers.Dense(512, activation = "relu"))
modelo1.add(keras.layers.Dense(256, activation = "relu"))
modelo1.add(keras.layers.Dense(64, activation = "relu"))
modelo1.add(keras.layers.Dense(1, activation = "sigmoid"))

treinar_modelo(modelo1, treino_x, treino_y)
print("Modelo 1 =================")
modelo1_train_predictions = (modelo1.predict(treino_x) > 0.5).astype(int).reshape(-1)
train_accuracy = accuracy_score(treino_y.reshape(-1), modelo1_train_predictions)
print(f'\n\nAcurácia no treino: {train_accuracy}')
modelo1_test_predictions = (modelo1.predict(teste_x) > 0.5).astype(int).reshape(-1)
test_accuracy = accuracy_score(teste_y.reshape(-1), modelo1_test_predictions)
print(f'Acurácia no teste: {test_accuracy}')
### Fim do código para o Modelo 1 ###

### Início do código para o Modelo 2 ###
modelo2 = keras.Sequential()
modelo2.add(keras.Input(treino_x[0].shape))
modelo2.add(keras.layers.Dense(512, activation = "relu"))
modelo2.add(keras.layers.Dense(64, activation = "relu"))
modelo2.add(keras.layers.BatchNormalization())
```

```
modelo2.add(keras.layers.BatchNormalization())
modelo2.add(keras.layers.Dense(1, activation = "sigmoid"))
treinar_modelo(modelo2, treino_x, treino_y, learning_rate = 0.0001, otimizador = keras
print("\n\nModelo 2 =================")
modelo2_train_predictions = (modelo2.predict(treino_x) > 0.5).astype(int).reshape(-1)
train_accuracy = accuracy_score(treino_y.reshape(-1), modelo2_train_predictions)
print(f'\n\nAcurácia no treino: {train_accuracy}')
modelo2_test_predictions = (modelo2.predict(teste_x) > 0.5).astype(int).reshape(-1)
test_accuracy = accuracy_score(teste_y.reshape(-1), modelo2_test_predictions)
print(f'Acurácia no teste: {test_accuracy}')
### Fim do código para o Modelo 2 ###

```

Model: "sequential_4"

Layer (type)	Output Shape	Param
dense_12 (Dense)	(None, 512)	6,291,96
dense_13 (Dense)	(None, 256)	131,32
dense_14 (Dense)	(None, 64)	16,44
dense_15 (Dense)	(None, 1)	6

Total params: 6,439,809 (24.57 MB)
Trainable params: 6,439,809 (24.57 MB)
Non-trainable params: 0 (0.00 B)

```
Modelo 1 =================
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step

Acurácia no treino: 1.0
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
Acurácia no teste: 0.66
```

Model: "sequential_5"

Layer (type)	Output Shape	Param
dense_16 (Dense)	(None, 512)	6,291,96
dense_17 (Dense)	(None, 64)	32,83
batch_normalization_1 (BatchNormalization)	(None, 64)	25
dense_18 (Dense)	(None, 1)	6

Total params: 6,325,121 (24.13 MB)
Trainable params: 6,324,993 (24.13 MB)
Non-trainable params: 128 (512.00 B)

```
Modelo 2 =================
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
```

```
Acurácia no treino: 0.9569377990430622
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step
Acurácia no teste: 0.72
```

Analisando as redes treinadas (5pt)

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

```
Ambos os modelos obtiveram resultados parecidos, sendo que o segundo obteve um resultado de acurác
```

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.

```
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.

```
### Início do código ###
def print_metrics(models_pred_dict, metric):
    print(f"Metrica: {metric.__name__}")

    for key in models_pred_dict.keys():
        resultado = metric(teste_y[0], models_pred_dict[key])
        print(f"{key}: {resultado}")

    print("="*50, "\n")


models_pred_dict = {
```

```
    "m1": m1_test_predictions,
    "m2": m2_test_predictions,
    "m3": m3_test_predictions,
    "m4": m4_test_predictions,
    "modelo1": modelo1_test_predictions,
    "modelo2": modelo2_test_predictions,
}

print_metrics(models_pred_dict, accuracy_score)
print_metrics(models_pred_dict, f1_score)
print_metrics(models_pred_dict, precision_score)
print_metrics(models_pred_dict, recall_score)

### Fim do código ###
```

```
Metrica: accuracy_score
m1: 0.72
m2: 0.58
m3: 0.74
m4: 0.76
modelo1: 0.66
modelo2: 0.72
==================================================

Metrica: f1_score
m1: 0.78125
m2: 0.5714285714285714
m3: 0.7936507936507936
m4: 0.8181818181818182
modelo1: 0.7213114754098361
modelo2: 0.7878787878787878
==================================================

Metrica: precision_score
m1: 0.8064516129032258
m2: 0.875
m3: 0.8333333333333334
m4: 0.8181818181818182
modelo1: 0.7857142857142857
modelo2: 0.7878787878787878
==================================================

Metrica: recall_score
m1: 0.7575757575757576
m2: 0.42424242424242425
m3: 0.7575757575757576
m4: 0.8181818181818182
modelo1: 0.6666666666666666
modelo2: 0.7878787878787878
==================================================
```

Analisando o treinamento dos modelos

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

O modelo m4 (O primeiro para desenvolvimento livre pelo aluno) é o que apresenta uma melhor harmon

O modelo m2 tem a maior precisão, com um maior indíce de true-positives, dessa forma, dentre as pr

O modelo m4 é o que melhor identifica imagem que contém gatos ao evitar os falsos negativos com o