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.
- Utilização do *Stratified K-fold*.
- Cálculos de métircas

Data da entrega : 07/11

- 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.

- *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*.

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy

from sklearn.metrics import accuracy_score

from tensorflow import keras
```

Configurando os *plots* de gráficos

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

```
%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)
```

Configurando o Google Colab.

Configurando o Google Colab para acessar os nossos dados.

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

```
Mounted at /content/drive
```

Carregando e préprocessamento dos dados

```
# Função para ler os dados (gato/não-gato)
def load_dataset():
    def _load_data():
      train_dataset = h5py.File('drive/MyDrive/train_catvnoncat.h5', "r")
      train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train set features
      train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train set labels

      test_dataset = h5py.File('drive/MyDrive/test_catvnoncat.h5', "r")
      test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set features
      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, classes

    def _preprocess_dataset(_treino_x_orig, _teste_x_orig):
      # Formate o conjunto de treinamento e teste dados de treinamento e teste para que as imagens
      # 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: vetorizar os dados de treinamento
      _teste_x_vet = _teste_x_orig.reshape(_teste_x_orig.shape[0], -1) # ToDo: vetorizar os dados de teste aqui

      # 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 (85pt)

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

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).

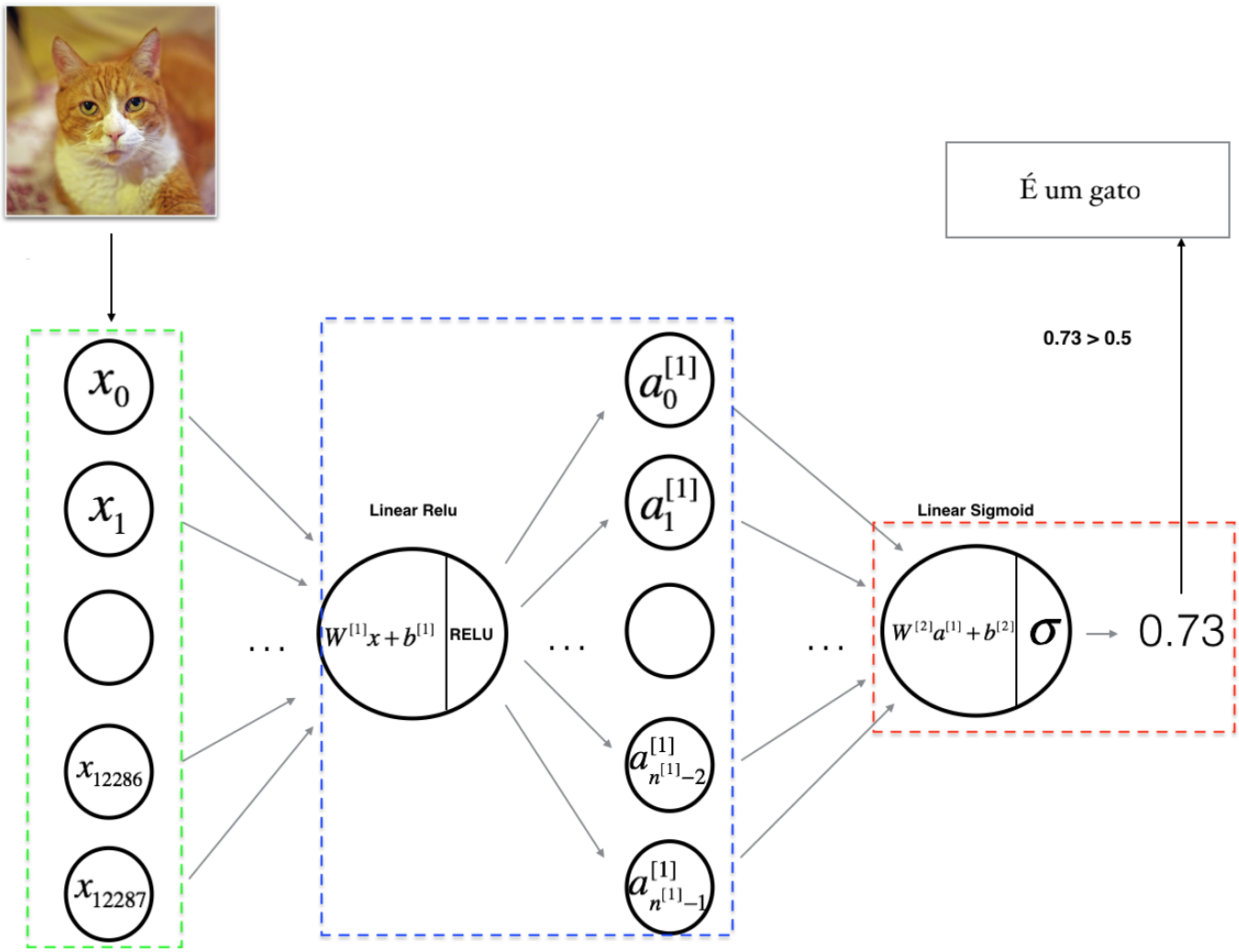


Figura 7: Rede neural com 2 camadas.

Resumo do modelo: ***ENTRADA -> LINEAR -> RELU -> LINEAR -> SIGMOID -> SAIDA***.

```
def treinar_modelo(modelo, x, y, epochs=100):
    # Setando a seed
    np.random.seed(10)

    # 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
```

ToDo: Definindo o modelo (5pt)

```
# Definição do modelo
def modelo_1():
  _model = keras.Sequential() # Crie um modelo sequencial com keras.Sequential
  _model.add(keras.layers.Dense(8,input_shape = (12288,), activation = 'relu')) # ToDo: Adicione uma camada densa c
  _model.add(keras.layers.Dense(1,activation = 'sigmoid')) # ToDo: Adicione uma camada densa com 1 neurônio e ativa
  return _model
```

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

ToDo: Instanciando o modelo e testando (5pt)

```
import tensorflow as tf
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 modelo
y_treino_pred =m1.predict(treino_x).reshape(1,-1)
y_teste_pred = m1.predict(teste_x).reshape(1,-1)

accuracy_treino = accuracy_score(treino_y.reshape(-1), np.round(y_treino_pred).astype(int).reshape(-1)) * 100
accuracy_teste = accuracy_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int).reshape(-1)) *100
## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcula
                                        # **dica** use o model.predict para predizer os dados e use o reshape com
print(f'Acurácia no teste: {accuracy_teste}') # ToDo: Utilize a função accuracy_score do sklearn para calcular a ac
                                    # **dica** use o model.predict para predizer os dados e use o reshape com -1 no
```

```
Epoch 90/100
7/7 [==============================] - 0s 6ms/step - loss: 0.6525 - accuracy: 0.6555
Epoch 91/100
7/7 [==============================] - 0s 4ms/step - loss: 0.6523 - accuracy: 0.6555
Epoch 92/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6521 - accuracy: 0.6555
Epoch 93/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6519 - accuracy: 0.6555
Epoch 94/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6517 - accuracy: 0.6555
Epoch 95/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6515 - accuracy: 0.6555
Epoch 96/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6513 - accuracy: 0.6555
Epoch 97/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6512 - accuracy: 0.6555
Epoch 98/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6510 - accuracy: 0.6555
Epoch 99/100
7/7 [==============================] - 0s 6ms/step - loss: 0.6508 - accuracy: 0.6555
Epoch 100/100
7/7 [==============================] - 0s 5ms/step - loss: 0.6507 - accuracy: 0.6555
7/7 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 6ms/step


Acurácia no treino: 65.55023923444976
Acurácia no teste: 34.0
```

Resultado esperado:

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

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

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

ToDo: Definição do modelo (10pt)

```
# Definição do modelo
def modelo_2():
  _model = keras.Sequential()
  _model.add(keras.layers.Dense(256,input_shape = (12288,), activation = 'relu')) # ToDo: Adicione uma camada densa
  _model.add(keras.layers.Dense(1,activation = 'sigmoid'))

  return _model
```

Agora treine e teste o seu modelo.

```
tf.random.set_seed(10)

# 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 modelo
y_treino_pred =m2.predict(treino_x).reshape(-1)
y_teste_pred = m2.predict(teste_x).reshape(-1)

accuracy_treino = accuracy_score(treino_y.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
accuracy_teste = accuracy_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int)) *100
```

```
## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcula
                                    # **dica** use o model.predict para predizer os dados e use o reshape com
print(f'Acurácia no teste: {accuracy_teste}') # ToDo: Utilize a função accuracy_score do sklearn para calcular a ac
```

```
Epoch 75/100
7/7 [==============================] - 0s 50ms/step - loss: 0.0222 - accuracy: 1.0000
Epoch 76/100
7/7 [==============================] - 0s 51ms/step - loss: 0.0205 - accuracy: 1.0000
Epoch 77/100
7/7 [==============================] - 0s 52ms/step - loss: 0.0209 - accuracy: 1.0000
Epoch 78/100
7/7 [==============================] - 0s 58ms/step - loss: 0.0204 - accuracy: 1.0000
Epoch 79/100
7/7 [==============================] - 0s 57ms/step - loss: 0.0189 - accuracy: 1.0000
Epoch 80/100
7/7 [==============================] - 0s 49ms/step - loss: 0.0205 - accuracy: 1.0000
Epoch 81/100
7/7 [==============================] - 0s 52ms/step - loss: 0.0189 - accuracy: 1.0000
Epoch 82/100
7/7 [==============================] - 0s 52ms/step - loss: 0.0182 - accuracy: 1.0000
Epoch 83/100
7/7 [==============================] - 0s 54ms/step - loss: 0.0172 - accuracy: 1.0000
Epoch 84/100
7/7 [==============================] - 0s 43ms/step - loss: 0.0176 - accuracy: 1.0000
Epoch 85/100
7/7 [==============================] - 0s 37ms/step - loss: 0.0173 - accuracy: 1.0000
Epoch 86/100
7/7 [==============================] - 0s 36ms/step - loss: 0.0183 - accuracy: 1.0000
Epoch 87/100
7/7 [==============================] - 0s 36ms/step - loss: 0.0200 - accuracy: 1.0000
Epoch 88/100
7/7 [==============================] - 0s 36ms/step - loss: 0.0137 - accuracy: 1.0000
Epoch 89/100
7/7 [==============================] - 0s 37ms/step - loss: 0.0135 - accuracy: 1.0000
Epoch 90/100
7/7 [==============================] - 0s 39ms/step - loss: 0.0142 - accuracy: 1.0000
Epoch 91/100
7/7 [==============================] - 0s 37ms/step - loss: 0.0135 - accuracy: 1.0000
Epoch 92/100
7/7 [==============================] - 0s 36ms/step - loss: 0.0138 - accuracy: 1.0000
Epoch 93/100
7/7 [==============================] - 0s 35ms/step - loss: 0.0145 - accuracy: 1.0000
Epoch 94/100
7/7 [==============================] - 0s 35ms/step - loss: 0.0127 - accuracy: 1.0000
Epoch 95/100
7/7 [==============================] - 0s 38ms/step - loss: 0.0127 - accuracy: 1.0000
Epoch 96/100
7/7 [==============================] - 0s 37ms/step - loss: 0.0110 - accuracy: 1.0000
Epoch 97/100
7/7 [==============================] - 0s 38ms/step - loss: 0.0108 - accuracy: 1.0000
Epoch 98/100
7/7 [==============================] - 0s 38ms/step - loss: 0.0104 - accuracy: 1.0000
Epoch 99/100
7/7 [==============================] - 0s 36ms/step - loss: 0.0107 - accuracy: 1.0000
Epoch 100/100
7/7 [==============================] - 0s 38ms/step - loss: 0.0105 - accuracy: 1.0000
7/7 [==============================] - 0s 7ms/step
2/2 [==============================] - 0s 10ms/step


Acurácia no treino: 100.0
Acurácia no teste: 76.0
```

Resultado esperado:

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

ToDo: Análise dos resultados (5pt)

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

```
Porque o segundo modelo e mais complexo e conseguiu generalizar melhor
```

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

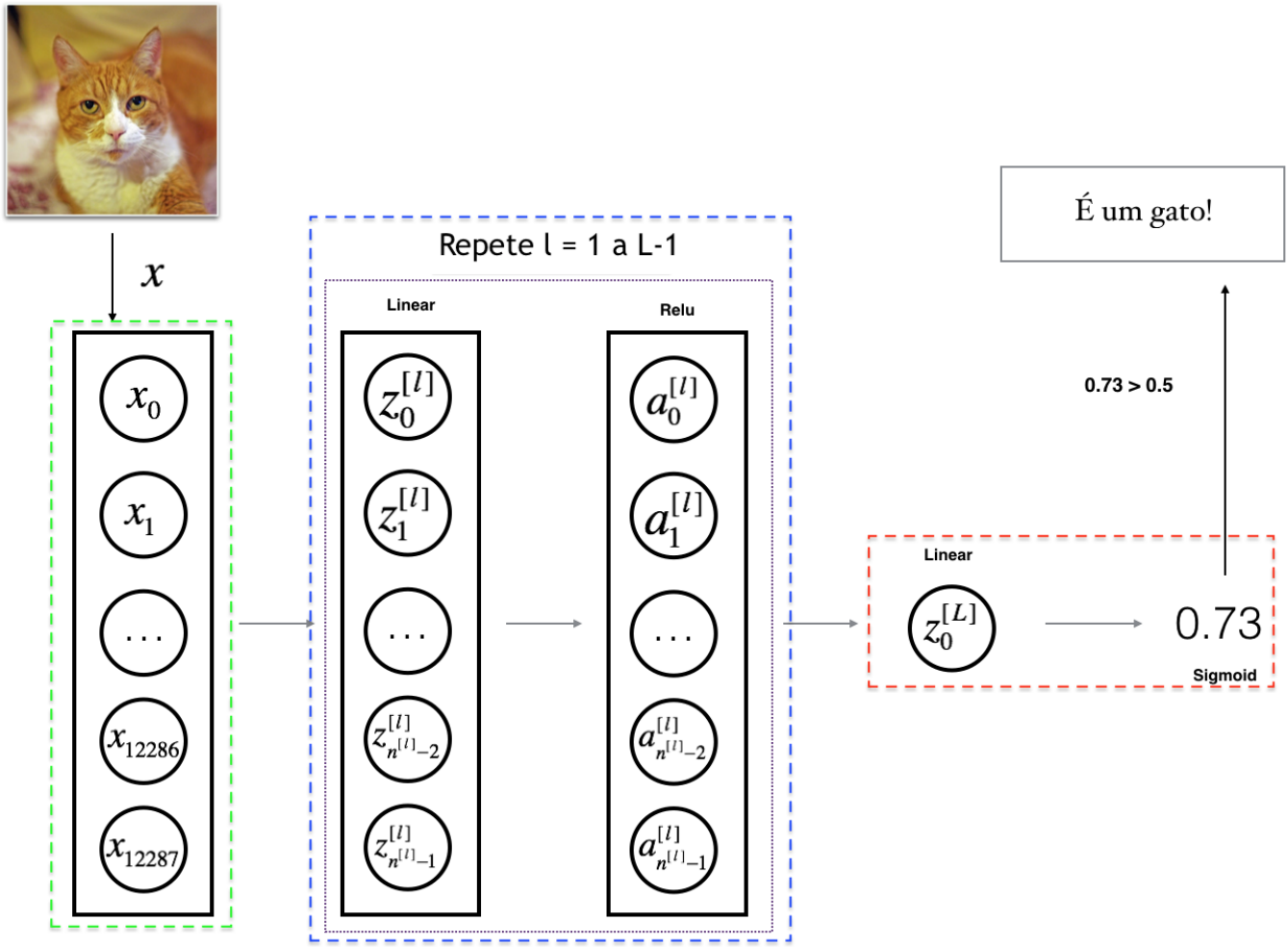


Figura 8: Rede neural com L camadas.
Resumo do modelo: ***ENTRADA -> LINEAR -> RELU -> LINEAR -> SIGMOID -> SAIDA***.

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.

ToDo: Definição do modelo (10pt)

```
# Definição do modelo
def modelo_3():
  _model = keras.Sequential()
  _model.add(keras.layers.Dense(256,input_shape = (12288,), activation = 'relu'))
  _model.add(keras.layers.Dense(64,input_shape = (12288,), activation = 'relu'))
  _model.add(keras.layers.Dense(8,input_shape = (12288,), activation = 'relu')) # ToDo: Adicione uma camada densa c
```

```
  _model.add(keras.layers.Dense(1,activation = 'sigmoid'))
  return _model
```

Agora treine e teste o seu modelo.

```
tf.random.set_seed(10)

# 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 modelo
y_treino_pred =m3.predict(treino_x).reshape(-1)
y_teste_pred = m3.predict(teste_x).reshape(-1)

accuracy_treino = accuracy_score(treino_y.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
accuracy_teste = accuracy_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int)) *100

## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcula
                                        # **dica** use o model.predict para predizer os dados e use o reshape com
print(f'Acurácia no teste: {accuracy_teste}') # ToDo: Utilize a função accuracy_score do sklearn para calcular a ac
```

```
7/7 [==============================] - 0s 37ms/step - loss: 0.0019 - accuracy: 1.0000
Epoch 100/100
7/7 [==============================] - 0s 35ms/step - loss: 0.0019 - accuracy: 1.0000
7/7 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step


Acurácia no treino: 100.0
Acurácia no teste: 70.0
```

Resultado esperado:

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

ToDo: Análise dos resultados (5pt)

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

```
Foi melhor, pois ele tem mais camadas que o outro, assim ele pode manipular os pesos melhor
```

Testando uma rede que você desenvolveu (15pt)

Crie uma arquitetura e treine/teste o seu modelo

ToDo: Definição do modelo (10pt)

```
# Definição do modelo
def meu_modelo():
    _model = keras.Sequential()
    _model.add(keras.layers.Dense(64,input_shape = (12288,), activation = 'relu')) # ToDo: Adicione uma camada dens
    _model.add(keras.layers.Dense(8,input_shape = (12288,), activation = 'relu'))
    _model.add(keras.layers.Dense(1,activation = 'sigmoid'))
    _model.add(keras.layers.Dropout(.2))

    return _model


tf.random.set_seed(10)

# 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 modelo
y_treino_pred =m4.predict(treino_x).reshape(-1)
y_teste_pred = m4.predict(teste_x).reshape(-1)

accuracy_treino = accuracy_score(treino_y.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
accuracy_teste = accuracy_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int)) *100

## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcula
                                      # **dica** use o model.predict para predizer os dados e use o reshape com
print(f'Acurácia no teste: {accuracy_teste}') # ToDo: Utilize a função accuracy_score do sklearn para calcular a ac
```

```
Epoch 77/100
7/7 [==============================] - 0s 12ms/step - loss: 1.5495 - accuracy: 0.8804
Epoch 78/100
7/7 [==============================] - 0s 12ms/step - loss: 1.2819 - accuracy: 0.8947
Epoch 79/100
7/7 [==============================] - 0s 11ms/step - loss: 1.5777 - accuracy: 0.8612
Epoch 80/100
7/7 [==============================] - 0s 12ms/step - loss: 1.4835 - accuracy: 0.8708
Epoch 81/100
7/7 [==============================] - 0s 12ms/step - loss: 1.2625 - accuracy: 0.8947
Epoch 82/100
7/7 [==============================] - 0s 11ms/step - loss: 1.1080 - accuracy: 0.9139
Epoch 83/100
7/7 [==============================] - 0s 11ms/step - loss: 1.5380 - accuracy: 0.8900
Epoch 84/100
7/7 [==============================] - 0s 12ms/step - loss: 1.2574 - accuracy: 0.9043
Epoch 85/100
7/7 [==============================] - 0s 13ms/step - loss: 1.3260 - accuracy: 0.8995
Epoch 86/100
7/7 [==============================] - 0s 11ms/step - loss: 1.6133 - accuracy: 0.8756
Epoch 87/100
7/7 [==============================] - 0s 12ms/step - loss: 0.8834 - accuracy: 0.9330
Epoch 88/100
7/7 [==============================] - 0s 12ms/step - loss: 0.9555 - accuracy: 0.9282
Epoch 89/100
7/7 [==============================] - 0s 11ms/step - loss: 0.9676 - accuracy: 0.9187
Epoch 90/100
7/7 [==============================] - 0s 12ms/step - loss: 1.2500 - accuracy: 0.9043
Epoch 91/100
7/7 [==============================] - 0s 12ms/step - loss: 1.3853 - accuracy: 0.8995
Epoch 92/100
7/7 [==============================] - 0s 12ms/step - loss: 1.4636 - accuracy: 0.8852
Epoch 93/100
7/7 [==============================] - 0s 12ms/step - loss: 1.1750 - accuracy: 0.9043
Epoch 94/100
7/7 [==============================] - 0s 13ms/step - loss: 1.3277 - accuracy: 0.8995
Epoch 95/100
7/7 [==============================] - 0s 12ms/step - loss: 0.8151 - accuracy: 0.9378
Epoch 96/100
7/7 [==============================] - 0s 13ms/step - loss: 1.5209 - accuracy: 0.8900
Epoch 97/100
7/7 [==============================] - 0s 12ms/step - loss: 0.8049 - accuracy: 0.9426
Epoch 98/100
7/7 [==============================] - 0s 12ms/step - loss: 0.8332 - accuracy: 0.9139
Epoch 99/100
7/7 [==============================] - 0s 12ms/step - loss: 1.3319 - accuracy: 0.8900
Epoch 100/100
7/7 [==============================] - 0s 11ms/step - loss: 1.5342 - accuracy: 0.8804
7/7 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 7ms/step


Acurácia no treino: 96.17224880382776
Acurácia no teste: 64.0
```

ToDo: Análise dos resultados (5pt)

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

```
ele obteve os mesmo resultados
```

Variando alguns hiperparâmetros (20pt)

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).

Você criar uma nova função para treinamento ou adaptar a existente.

ToDo: Desenvolva os seus modelos aqui (15pt)

```
def treinar_modelo1(modelo, x, y, epochs=100):
    # Setando a seed
    np.random.seed(10)

    # Compilando o modelo
    modelo.compile(optimizer='adadelta',
                   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


def treinar_modelo2(modelo, x, y, epochs=100):
    # Setando a seed
    np.random.seed(10)

    # Compilando o modelo
    modelo.compile(optimizer='rmsprop',
                   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


### Início do código ###
def meu_modelo5():
    _model = keras.Sequential()
    _model.add(keras.layers.Dense(16,input_shape = (12288,), activation = 'gelu')) # ToDo: Adicione uma camada dens
    _model.add(keras.layers.Dense(8,input_shape = (12288,), activation = 'gelu'))
    _model.add(keras.layers.Dense(1,activation = 'sigmoid'))
    _model.add(keras.layers.Dropout(.2))
    return _model
### Fim do código ###


tf.random.set_seed(1)

# Criando o modelo
m5 = meu_modelo5() # ToDo: chame a função que define o modelo

# Treinando o modelo
m5 = treinar_modelo1(m5,treino_x,treino_y) # ToDo: Chame a função para treinar o modelo
y_treino_pred =m5.predict(treino_x).reshape(-1)
y_teste_pred = m5.predict(teste_x).reshape(-1)

accuracy_treino = accuracy_score(treino_y.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
accuracy_teste = accuracy_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int)) *100

## Predição da rede
```

```
print(f'\n\nAcurácia no treino: {accuracy_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcula
                                        # **dica** use o model.predict para predizer os dados e use o reshape com
print(f'Acurácia no teste: {accuracy_teste}')
```

```
Epoch 75/100
7/7 [==============================] - 0s 6ms/step - loss: 1.5454 - accuracy: 0.6507
Epoch 76/100
7/7 [==============================] - 0s 6ms/step - loss: 1.8151 - accuracy: 0.6459
Epoch 77/100
7/7 [==============================] - 0s 6ms/step - loss: 1.6045 - accuracy: 0.6603
Epoch 78/100
7/7 [==============================] - 0s 6ms/step - loss: 1.2429 - accuracy: 0.6699
Epoch 79/100
7/7 [==============================] - 0s 7ms/step - loss: 1.1874 - accuracy: 0.6699
Epoch 80/100
7/7 [==============================] - 0s 6ms/step - loss: 1.4495 - accuracy: 0.6651
Epoch 81/100
7/7 [==============================] - 0s 6ms/step - loss: 1.5519 - accuracy: 0.6603
Epoch 82/100
7/7 [==============================] - 0s 6ms/step - loss: 1.6474 - accuracy: 0.6699
Epoch 83/100
7/7 [==============================] - 0s 6ms/step - loss: 1.5979 - accuracy: 0.6603
Epoch 84/100
7/7 [==============================] - 0s 6ms/step - loss: 1.5435 - accuracy: 0.6603
Epoch 85/100
7/7 [==============================] - 0s 6ms/step - loss: 1.4010 - accuracy: 0.6364
Epoch 86/100
7/7 [==============================] - 0s 6ms/step - loss: 1.2929 - accuracy: 0.6699
Epoch 87/100
7/7 [==============================] - 0s 6ms/step - loss: 1.7206 - accuracy: 0.6651
Epoch 88/100
7/7 [==============================] - 0s 11ms/step - loss: 1.7489 - accuracy: 0.6459
Epoch 89/100
7/7 [==============================] - 0s 9ms/step - loss: 2.0177 - accuracy: 0.6555
Epoch 90/100
7/7 [==============================] - 0s 10ms/step - loss: 1.5317 - accuracy: 0.6555
Epoch 91/100
7/7 [==============================] - 0s 9ms/step - loss: 1.4555 - accuracy: 0.6555
Epoch 92/100
7/7 [==============================] - 0s 10ms/step - loss: 1.2469 - accuracy: 0.6507
Epoch 93/100
7/7 [==============================] - 0s 7ms/step - loss: 1.3859 - accuracy: 0.6699
Epoch 94/100
7/7 [==============================] - 0s 8ms/step - loss: 1.3883 - accuracy: 0.6651
Epoch 95/100
7/7 [==============================] - 0s 10ms/step - loss: 1.9404 - accuracy: 0.6507
Epoch 96/100
7/7 [==============================] - 0s 12ms/step - loss: 1.3885 - accuracy: 0.6507
Epoch 97/100
7/7 [==============================] - 0s 10ms/step - loss: 1.9587 - accuracy: 0.6364
Epoch 98/100
7/7 [==============================] - 0s 9ms/step - loss: 1.1846 - accuracy: 0.6507
Epoch 99/100
7/7 [==============================] - 0s 9ms/step - loss: 1.8567 - accuracy: 0.6555
Epoch 100/100
7/7 [==============================] - 0s 9ms/step - loss: 1.3939 - accuracy: 0.6364
7/7 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 6ms/step


Acurácia no treino: 64.5933014354067
Acurácia no teste: 36.0
```

```
def meu_modelo6():
    _model = keras.Sequential()
    _model.add(keras.layers.Dense(8,input_shape = (12288,), activation = keras.layers.LeakyReLU(alpha = 0.01)))
    _model.add(keras.layers.Dense(1,activation = 'sigmoid'))
    _model.add(keras.layers.Dropout(.2))
    return _model


tf.random.set_seed(1)

# Criando o modelo
```

```
m6 = meu_modelo6() # ToDo: chame a função que define o modelo

# Treinando o modelo
m6 = treinar_modelo2(m6,treino_x,treino_y) # ToDo: Chame a função para treinar o modelo
y_treino_pred = m6.predict(treino_x).reshape(-1)
y_teste_pred = m6.predict(teste_x).reshape(-1)

accuracy_treino = accuracy_score(treino_y.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
accuracy_teste = accuracy_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int)) *100

## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcula
                                        # **dica** use o model.predict para predizer os dados e use o reshape com
print(f'Acurácia no teste: {accuracy_teste}')
```

```
Epoch 75/100
7/7 [==============================] - 0s 6ms/step - loss: 1.6973 - accuracy: 0.7273
Epoch 76/100
7/7 [==============================] - 0s 7ms/step - loss: 1.6050 - accuracy: 0.7799
Epoch 77/100
7/7 [==============================] - 0s 9ms/step - loss: 1.3619 - accuracy: 0.8230
Epoch 78/100
7/7 [==============================] - 0s 7ms/step - loss: 1.0972 - accuracy: 0.7703
Epoch 79/100
7/7 [==============================] - 0s 6ms/step - loss: 0.9412 - accuracy: 0.8421
Epoch 80/100
7/7 [==============================] - 0s 6ms/step - loss: 1.2370 - accuracy: 0.8038
Epoch 81/100
7/7 [==============================] - 0s 6ms/step - loss: 1.3867 - accuracy: 0.7751
Epoch 82/100
7/7 [==============================] - 0s 6ms/step - loss: 1.4531 - accuracy: 0.8038
Epoch 83/100
7/7 [==============================] - 0s 7ms/step - loss: 2.4920 - accuracy: 0.6555
Epoch 84/100
7/7 [==============================] - 0s 6ms/step - loss: 1.3055 - accuracy: 0.7847
Epoch 85/100
7/7 [==============================] - 0s 6ms/step - loss: 1.1364 - accuracy: 0.8565
Epoch 86/100
7/7 [==============================] - 0s 7ms/step - loss: 1.0024 - accuracy: 0.8517
Epoch 87/100
7/7 [==============================] - 0s 7ms/step - loss: 1.5772 - accuracy: 0.8278
Epoch 88/100
7/7 [==============================] - 0s 6ms/step - loss: 1.4289 - accuracy: 0.8134
Epoch 89/100
7/7 [==============================] - 0s 6ms/step - loss: 1.7885 - accuracy: 0.8134
Epoch 90/100
7/7 [==============================] - 0s 7ms/step - loss: 1.4596 - accuracy: 0.7656
Epoch 91/100
7/7 [==============================] - 0s 6ms/step - loss: 1.2084 - accuracy: 0.8325
Epoch 92/100
7/7 [==============================] - 0s 6ms/step - loss: 0.9985 - accuracy: 0.8182
Epoch 93/100
7/7 [==============================] - 0s 6ms/step - loss: 1.0957 - accuracy: 0.8565
Epoch 94/100
7/7 [==============================] - 0s 6ms/step - loss: 1.4243 - accuracy: 0.7847
Epoch 95/100
7/7 [==============================] - 0s 6ms/step - loss: 1.7028 - accuracy: 0.8230
Epoch 96/100
7/7 [==============================] - 0s 6ms/step - loss: 1.2131 - accuracy: 0.7560
Epoch 97/100
7/7 [==============================] - 0s 8ms/step - loss: 1.7005 - accuracy: 0.7943
Epoch 98/100
7/7 [==============================] - 0s 6ms/step - loss: 0.9221 - accuracy: 0.8373
Epoch 99/100
7/7 [==============================] - 0s 6ms/step - loss: 1.5924 - accuracy: 0.8325
Epoch 100/100
7/7 [==============================] - 0s 6ms/step - loss: 1.5583 - accuracy: 0.7751
7/7 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 6ms/step


Acurácia no treino: 90.43062200956938
Acurácia no teste: 70.0
```

ToDo: Analisando redes treinadas (5pt)

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

```
 o com menos camadas, pois as vezes ter mais camadas nem sempre vai fazer vc ter resultados melhores
```

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 os modelos somente nos dados de teste.

```
### Início do código ###
def meu_modelo7():
    _model = keras.Sequential()
    _model.add(keras.layers.Dense(8,input_shape = (12288,), activation = keras.layers.LeakyReLU(alpha = 0.01)))
    _model.add(keras.layers.Dense(1,activation = 'sigmoid'))
    _model.add(keras.layers.Dropout(.2))
    return _model
## Fim do código ###

tf.random.set_seed(10)

# Criando o modelo
m6 = meu_modelo7() # ToDo: chame a função que define o modelo

# Treinando o modelo
m6 = treinar_modelo2(m6,treino_x,treino_y) # ToDo: Chame a função para treinar o modelo
y_treino_pred = m6.predict(treino_x).reshape(-1)
y_teste_pred = m6.predict(teste_x).reshape(-1)

accuracy_treino = accuracy_score(treino_y.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
accuracy_teste = accuracy_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int)) *100

## Predição da rede
# print(f'\n\nAcurácia no treino: {accuracy_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcu

# print(f'Acurácia no teste: {accuracy_teste}')

###--------------------------------------------------------------------------------###

# Criando o modelo
m7 = meu_modelo7() # ToDo: chame a função que define o modelo

# Treinando o modelo
m7 = treinar_modelo2(m7,treino_x,treino_y)
y_treino_pred = m7.predict(treino_x).reshape(-1)
y_teste_pred = m7.predict(teste_x).reshape(-1)
```

```
f1_treino = f1_score(treino_y.reshape(-1),np.round(y_treino_pred).astype(int), average='macro') *100
f1_teste = f1_score(teste_y.reshape(-1), np.round(y_teste_pred).astype(int), average='macro')  *100

# ## Predição da rede
# print(f'\n\nMétrica F1 no treino: {f1_treino}')
# print(f'Métrica F1 no teste: {f1_teste}')

###-------------------------------------------------------------------------------------###


# Criando o modelo
m8 = meu_modelo7() # ToDo: chame a função que define o modelo

# Treinando o modelo
m8 = treinar_modelo2(m8,treino_x,treino_y) # ToDo: Chame a função para treinar o modelo
y_treino_pred = m8.predict(treino_x).reshape(-1)
y_teste_pred = m8.predict(teste_x).reshape(-1)

score_treino = precision_score(treino_y.reshape(-1),  np.round(y_treino_pred).astype(int), average='macro') * 100
score_teste = precision_score(teste_y.reshape(-1),  np.round(y_teste_pred).astype(int), average='macro') *100

## Predição da rede
# print(f'\n\nAcurácia no treino: {score_treino}') # ToDo: Utilize a função accuracy_score do sklearn para calcular
#                                         # **dica** use o model.predict para predizer os dados e use o reshape co
# print(f'Acurácia no teste: {score_teste}')



###-------------------------------------------------------------------------------------###

m9 = meu_modelo7() # ToDo: chame a função que define o modelo

# Treinando o modelo
m9 = treinar_modelo2(m9,treino_x,treino_y) # ToDo: Chame a função para treinar o modelo
y_treino_pred = m9.predict(treino_x).reshape(-1)
y_teste_pred = m9.predict(teste_x).reshape(-1)

recall_treino = recall_score(treino_y.reshape(-1),  np.round(y_treino_pred).astype(int), average='macro') * 100
recall_teste = recall_score(teste_y.reshape(-1),  np.round(y_teste_pred).astype(int), average='macro') *100


## Predição da rede
print(f'\n\nAcurácia no treino: {accuracy_treino}')
print(f'Acurácia no teste: {accuracy_teste}')
print(f'\n\nMétrica F1 no treino: {f1_treino}')
print(f'Métrica F1 no teste: {f1_teste}')
print(f'\n\nprecision_score no treino: {score_treino}')
print(f'precision_score no teste: {score_teste}')
print(f'\n\nrecall_score no treino: {recall_treino }')
print(f'recall_score no teste: {recall_teste }')
```

```
[==============================] - 0s 7ms/step - loss: 1.5339 - accuracy: 0.8182
h 92/100
[==============================] - 0s 6ms/step - loss: 1.5438 - accuracy: 0.8421
h 93/100
[==============================] - 0s 6ms/step - loss: 1.2379 - accuracy: 0.8373
h 94/100
[==============================] - 0s 7ms/step - loss: 2.3071 - accuracy: 0.7656
h 95/100
[==============================] - 0s 7ms/step - loss: 1.2027 - accuracy: 0.8517
h 96/100
[==============================] - 0s 7ms/step - loss: 1.6761 - accuracy: 0.8182
h 97/100
[==============================] - 0s 8ms/step - loss: 0.8592 - accuracy: 0.8612
h 98/100
[==============================] - 0s 7ms/step - loss: 1.0034 - accuracy: 0.8278
h 99/100
[==============================] - 0s 6ms/step - loss: 1.6830 - accuracy: 0.8565
h 100/100
[==============================] - 0s 6ms/step - loss: 2.0595 - accuracy: 0.7560
[==============================] - 0s 4ms/step
[==============================] - 0s 5ms/step


ácia no treino: 88.51674641148325
ácia no teste: 54.0


ica F1 no treino: 75.32467532467531
ica F1 no teste: 53.98159263705482


ision_score no treino: 17.22488038277512
ision_score no teste: 33.0


ll_score no treino: 87.39355231143553
ll_score no teste: 63.903743315508024
```

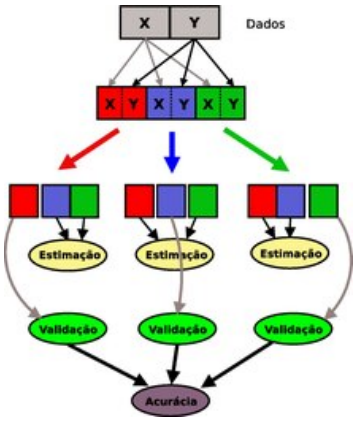
ToDo: O que você pode falar sobre os modelos treinados (5pt)

que os modelos agora tiveram um score no teste melhor que anteriormente

K-Fold (15pt)

O método de validação cruzada denominado *k-fold* consiste em dividir o conjunto total de dados em *k* subconjuntos mutuamente exclusivos do mesmo tamanho e, a partir daí, um subconjunto é utilizado para teste e os *k-1* restantes são utilizados para estimação dos parâmetros, fazendo-se o cálculo da acurácia do modelo.

A figura abaixo exemplifica um *3-fold*.



O *K-Fold* padrão divide nossos dados em *k* conjuntos sem prestar atenção no balanceamento dos dados, o que pode ocasionar com o que o seu modelo seja treinado somente com dados de uma classe e quando for testar, somente os dados da outra classe será usado, por exemplo. O *Stratified K-Fold* é uma alternativa, uma vez que faz a mesma coisa que o *K-Fold* mas com uma grande melhoria: obedece ao balanceamento (distribuição) dos labels.

ToDo: Avaliando o *Stratified K-Fold* (10pt)

Escolha um dos modelos treinados e o aplique a estratégia do *Stratified K-Fold* usando somente os *dados de treino* e *k = 3*. Reporte as métricas de acurácia, precisão, revocação e F1-score para cada **K** e também a média com desvio padrão geral.

Dicas:

- Utilize o *StratifiedKFold* presente na biblioteca *sklearn.model_selection*.
- Você pode ter problemas de memória se seu modelo você for muito grande, por isso considere o uso do comando *del* do python.
- Adapte o exemplo deste link para o problema dos gatos.
- Utilize somente os dados de treino aqui.

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=3)

m_acuracia_treino = []
m_f1_score_treino = []
m_recall_score_treino = []
m_precision_score_treino = []


m_acuracia_teste = []
m_f1_score_teste = []
m_recall_score_teste = []
m_precision_score_teste = []



for treino_index, teste_index in skf.split(treino_x, treino_y.reshape(-1)):
  x_treino_fold = treino_x[treino_index]
  y_treino_fold = treino_y.reshape(-1)[treino_index]

  x_teste_fold = treino_x[teste_index]
  y_teste_fold = treino_y.reshape(-1)[teste_index]

  modelo = modelo_3()

  modelo= treinar_modelo(modelo, x_treino_fold, y_treino_fold)

  y_treino_pred = modelo.predict(x_treino_fold).reshape(-1)
  y_teste_pred = modelo.predict(x_teste_fold).reshape(-1)

  acuracia_treino = accuracy_score(y_treino_fold.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
  f1_score_treino = f1_score(y_treino_fold.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
  recall_score_treino = f1_score(y_treino_fold.reshape(-1), np.round(y_treino_pred).astype(int)) * 100
  precision_score_treino = precision_score(y_treino_fold.reshape(-1), np.round(y_treino_pred).astype(int)) * 100

  acuracia_teste = accuracy_score(y_teste_fold.reshape(-1), np.round(y_teste_pred).astype(int)) * 100
  f1_score_teste = f1_score(y_teste_fold.reshape(-1), np.round(y_teste_pred).astype(int)) * 100
  recall_score_teste = f1_score(y_teste_fold.reshape(-1), np.round(y_teste_pred).astype(int)) * 100
  precision_score_teste = precision_score(y_teste_fold.reshape(-1), np.round(y_teste_pred).astype(int)) * 100


  print(f'Acurácia no treino: {acuracia_treino}')
  print(f'F1_Score no treino: {f1_score_treino}')
  print(f'Recall_Score no treino: {recall_score_treino}')
```

```
      print(f'precision_Score no treino: {precision_score_treino}')

      print(f'Acurácia no teste: {acuracia_teste}')
      print(f'F1_Score no teste: {f1_score_teste}')
      print(f'Recall_Score no teste: {recall_score_teste}')
      print(f'precision_Score no teste: {precision_score_teste}')


      m_acuracia_treino.append(acuracia_treino)
      m_f1_score_treino.append(f1_score_treino)
      m_recall_score_treino.append(recall_score_treino)
      m_precision_score_treino.append(precision_score_treino)

      m_acuracia_teste.append(acuracia_teste)
      m_f1_score_teste.append(f1_score_teste)
      m_recall_score_teste.append(recall_score_teste)
      m_precision_score_teste.append(precision_score_teste)

    print(f'Media da Acurácia no treino: {np.mean(m_acuracia_treino)}')
    print(f'Media do F1_Score no treino: {np.mean(m_f1_score_treino)}')
    print(f'Media do Recall_Score no treino: {np.mean(m_recall_score_treino)}')
    print(f'Media do precision_Score no treino: {np.mean(m_precision_score_treino)}')

    print(f'Media da Acurácia no teste: {np.mean(m_acuracia_teste)}')
    print(f'Media do F1_Score no teste: {np.mean(m_f1_score_teste)}')
    print(f'Media do Recall_Score no teste: {np.mean(m_recall_score_teste)}')
    print(f'Media do precision_Score no teste: {np.mean(m_precision_score_teste)}')


    print(f'Desvio Padrao da Acurácia no treino: {np.std(m_acuracia_treino)}')
    print(f'Desvio Padrao do F1_Score no treino: {np.std(m_f1_score_treino)}')
    print(f'Desvio Padrao do Recall_Score no treino: {np.std(m_recall_score_treino)}')
    print(f'Desvio Padrao do precision_Score no treino: {np.std(m_precision_score_treino)}')

    print(f'Desvio Padrao da Acurácia no teste: {np.std(m_acuracia_teste)}')
    print(f'Desvio Padrao do F1_Score no teste: {np.std(m_f1_score_teste)}')
    print(f'Desvio Padrao do Recall_Score no teste: {np.std(m_recall_score_teste)}')
    print(f'Desvio Padrao do precision_Score no teste: {np.std(m_precision_score_teste)}')
```

```
3/3 [==============================] - 0s 8ms/step
Acurácia no treino: 95.71428571428572
F1_Score no treino: 94.11764705882352
Recall_Score no treino: 94.11764705882352
precision_Score no treino: 88.88888888888889
Acurácia no teste: 95.65217391304348
F1_Score no teste: 94.11764705882352
Recall_Score no teste: 94.11764705882352
precision_Score no teste: 88.88888888888889
Media da Acurácia no treino: 83.22370674888661
Media do F1_Score no treino: 59.94397759103641
Media do Recall_Score no treino: 59.94397759103641
Media do precision_Score no treino: 54.629629629629626
Media da Acurácia no teste: 84.74120082815735
Media do F1_Score no teste: 61.561228264890865
Media do Recall_Score no teste: 61.561228264890865
Media do precision_Score no teste: 57.215836526181356
Desvio Padrao da Acurácia no treino: 12.897266895499003
Desvio Padrao do F1_Score no treino: 42.525399920108335
Desvio Padrao do Recall_Score no treino: 42.525399920108335
Desvio Padrao do precision_Score no treino: 39.04290489316299
Desvio Padrao da Acurácia no teste: 13.502362107268986
Desvio Padrao do F1_Score no teste: 43.55450303016714
Desvio Padrao do Recall_Score no teste: 43.55450303016714
Desvio Padrao do precision_Score no teste: 40.5350384160968
```

ToDo: Entendendo o *K-fold*.

Por que o *K-fold* pode ser uma estratégia mais robusta de análise do que a simples classificação ou divisão 80-20 dos dados (80% para treino e 20% para teste)? (5pt)

por que ele usa todo mundo pelo menos uma vez, assim tendo mais dificuldade de se ter overfitting