#Importação da biblioteca Pandas

import pandas as pd

#Definição das colunas e da base de dados

nome\_colunas=['sepal length', 'sepal width', 'petal length', 'petal width', 'Tipo']

banco\_de\_dados = pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', names=nome\_colunas)

#Definindo a saída como a coluna Tipo

saida = banco\_de\_dados[['Tipo']]

#Separando a saída em três colunas a serem preenchidas de forma booleana

saida = pd.get\_dummies(saida, columns=['Tipo'])

values = list(saida.columns.values)

saida.head()

#Retirada da coluna de saída do grupo de dados da entrada

rede = banco\_de\_dados.drop(columns=['Tipo'])

#Importação de função e divisão entre treino e teste

from sklearn.model\_selection import train\_test\_split

rede\_treino, rede\_teste, saida\_treino, saida\_teste = train\_test\_split(rede, saida,test\_size=0.2)

print("\nRede de Treino (sem a saída):\n")

print(rede\_treino.head())

print(rede\_treino.shape)

print("\nRede de Teste (sem a saída):\n")

print(rede\_teste.head())

print(rede\_teste.shape)

#Importação do modelo Sequencial e da forma densa para camadas

from keras.models import Sequential

from keras.layers import Dense

#Criação do modelo da rede

model = Sequential()

#Definição de variável armazenando o número de colunas

numero\_colunas = rede.shape[1]

#Adicionando as camadas

model.add(Dense(10, activation='relu', input\_shape=(numero\_colunas,)))

model.add(Dense(20, activation='relu'))

model.add(Dense(10, activation='softmax'))

model.add(Dense(3))

#Compilação do modelo e definição da optimização e função de perda

model.compile(optimizer='adam', loss='mean\_squared\_error')

#Interrompe as interações quando a rede deixa de se aprimorar

from keras.callbacks import EarlyStopping

early\_stopping\_monitor = EarlyStopping(patience=3)

#Treino do modelo

teste = model.fit(rede\_treino, saida\_treino, validation\_data=(rede\_teste, saida\_teste),  epochs=150, callbacks=[early\_stopping\_monitor])

import matplotlib.pyplot as plt

fig, graf\_perda = plt.subplots()

graf\_perda.plot(teste.history['loss'], 'r', marker ='.', label='Perda no Treino')

graf\_perda.plot(teste.history['val\_loss'], 'b', marker = '.', label='Perda na Validação')

graf\_perda.legend()

model.predict(rede\_treino.iloc[0:5])

#Importação da matriz de confusão

import numpy as np

from sklearn.metrics import confusion\_matrix

valor\_pred = np.zeros(rede\_treino.shape[0])

#Fazendo a predição da rede

predicao = model.predict(rede\_treino)

valor\_real = np.array(saida\_treino).argmax(axis=1)

for i in range(rede\_treino.shape[0]):

  valor\_pred[i] = predicao[i].argmax()

#Imprimindo a matroz

confusion\_matrix(valor\_real, valor\_pred)

#Calculando a acurária do algoritmo

from sklearn.metrics import accuracy\_score

acuracia = (accuracy\_score(valor\_real, valor\_pred) \* 100)

print ("A acurária dos casos de treino foi de %d" % (acuracia),"%.")

#Importação da matriz de confusão

import numpy as np

from sklearn.metrics import confusion\_matrix

valor\_pred = np.zeros(rede\_teste.shape[0])

#Fazendo a predição da rede

predicao = model.predict(rede\_teste)

valor\_real = np.array(saida\_teste).argmax(axis=1)

for i in range(rede\_teste.shape[0]):

  valor\_pred[i] = predicao[i].argmax()

#Imprimindo a matriz

confusion\_matrix(valor\_real, valor\_pred)

#Calculando a acurária do algoritmo

from sklearn.metrics import accuracy\_score

acuracia = (accuracy\_score(valor\_real, valor\_pred) \* 100)

print ("A acurária dos casos de teste foi de %d" % (acuracia),"%.")