**PSI3471 - Fundamentos de Sistemas Eletrônicos Inteligentes**

Lição de Casa 5 (Aulas 9 e 10)

| **Nome Completo** | **Número USP** |
| --- | --- |
| Gustavo Henrique da Silva Amaral | 12551686 |
| Thiago da Rocha Calomino Gonçalves | 12554647 |

**Lição de casa #5 (Parte 1)**

Escreva o algoritmo vizinho mais próximo pela força bruta manualmente (sem usar uma rotina pronta de OpenCV ou de alguma outra biblioteca) para classificar MNIST e compare o seu tempo de processamento com o tempo de processamento das funções do OpenCV.

**Explicação completa:**

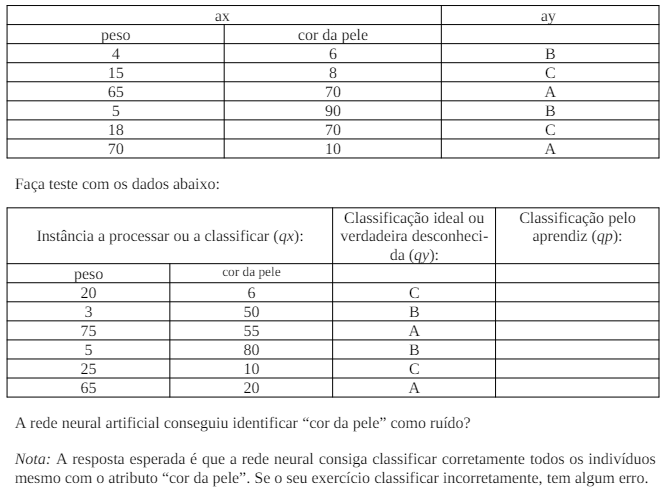
-

**Código da lição:**

| // Lição de casa 5 (Parte 1)  //  // Gustavo Henrique da Silva Amaral - 12551686  // Thiago da Rocha Calomino Gonçalves - 12554647  // Tem uma versão no Collab também  import numpy as np  from tensorflow.keras.datasets import mnist  from sklearn.model\_selection import train\_test\_split  import time  import cv2  ####### Importação do dataset  # Carregar o dataset MNIST  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # Redimensionar os dados para 2D (cada imagem 28x28 será um vetor de 784)  x\_train = x\_train.reshape(x\_train.shape[0], -1)  x\_test = x\_test.reshape(x\_test.shape[0], -1)  # Normalizar os dados  x\_train = x\_train.astype('float32') / 255  x\_test = x\_test.astype('float32') / 255  # Dividir os dados de teste para ter um conjunto menor para a comparação de tempo  x\_test, x\_test\_small, y\_test, y\_test\_small = train\_test\_split(x\_test, y\_test, test\_size=0.1, random\_state=42)  ## Dataset usado será menor para treino e teste em ambos os casos  ## para que o tempo não seja tão grande  ####### Implementação manual do KNN  # Reduzir o tamanho dos dados de treinamento e teste para acelerar a execução  small\_train\_size = 1000 # número menor de dados de treinamento  small\_test\_size = 100 # número menor de dados de teste  x\_train\_small = x\_train[:small\_train\_size]  y\_train\_small = y\_train[:small\_train\_size]  x\_test\_small = x\_test[:small\_test\_size]  y\_test\_small = y\_test[:small\_test\_size]  def euclidean\_distance(x1, x2):  return np.sqrt(np.sum((x1 - x2) \*\* 2))  def knn\_manual(x\_train, y\_train, x\_test, k=3):  y\_pred = []  for x in x\_test:  distances = []  for i in range(len(x\_train)):  dist = euclidean\_distance(x, x\_train[i])  distances.append((dist, y\_train[i]))  distances.sort(key=lambda x: x[0])  neighbors = [distances[i][1] for i in range(k)]  y\_pred.append(np.argmax(np.bincount(neighbors)))  return np.array(y\_pred)  # Medir o tempo de processamento do KNN manual com conjunto menor  start\_time = time.time()  y\_pred\_manual = knn\_manual(x\_train\_small, y\_train\_small, x\_test\_small, k=3)  manual\_time = time.time() - start\_time  ####### Implementação do KNN via OpenCV  # Treinar o modelo KNN usando OpenCV com conjunto menor  knn = cv2.ml.KNearest\_create()  knn.train(x\_train\_small.astype(np.float32), cv2.ml.ROW\_SAMPLE, y\_train\_small.astype(np.float32))  # Medir o tempo de processamento do KNN com OpenCV com conjunto menor  start\_time = time.time()  ret, result, neighbours, dist = knn.findNearest(x\_test\_small.astype(np.float32), k=3)  opencv\_time = time.time() - start\_time  ####### Tempos obtidos  ratio = manual\_time/opencv\_time  print(f"""  Tempo do KNN manual: {manual\_time:.4f} s  Tempo do KNN com OpenCV: {opencv\_time:.4f} s  O tempo manual é {ratio:.4f} vezes o tempo com OpenCV  """) |
| --- |

**Lição de casa #5 (Parte 2)**

Modifique o programa abc1.py para que aceite dois atributos na entrada: peso em kg e cor da pele (0 = escuro, 100 = claro). Treine a rede com os dados abaixo:



| # abc1.py  #abc1.py - 2024  import os; os.environ['TF\_CPP\_MIN\_LOG\_LEVEL']='3'  import tensorflow as tf  import tensorflow.keras as keras  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Activation  from tensorflow.keras import optimizers  from tensorflow.keras.utils import plot\_model  import numpy as np  import sys  model = Sequential()  model.add(Dense(3, activation='sigmoid', input\_dim=1))  model.add(Dense(3, activation='sigmoid'))  sgd=optimizers.SGD(learning\_rate=10);  model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])  ax = np.matrix('4; 15; 65; 5; 18; 70 ',dtype="float32")  ax=2\*(ax/100-0.5) #-1 a +1  ay = np.matrix('0 1 0; 0 0 1; 1 0 0; 0 1 0; 0 0 1; 1 0 0',dtype="float32")  model.fit(ax, ay, epochs=200, batch\_size=2, verbose=2)  qx = np.matrix('16; 3; 75 ',dtype="float32")  qx = 2 \* (qx/100-0.5) # -1 a +1  qy = np.matrix('0 0 1; 0 1 0; 1 0 0',dtype="float32")  teste = model.evaluate(qx,qy)  print(f"Custo e acuracidade de teste: {teste}")  qp2=model.predict(qx)  print(f"Classificacao de teste:\n {qp2}")  qp = qp2.argmax(axis=1)  print(f"Rotulo de saida:\n {qp}")  plot\_model(model, to\_file='abc1.png', show\_shapes=True)  model.summary() |
| --- |

Gpt resposta (abc1\_py → ) :

# abc1.py - 2024

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

import tensorflow as tf

import tensorflow.keras as keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras import optimizers

import numpy as np

# Definição do modelo

model = Sequential()

model.add(Dense(3, activation='sigmoid', input\_dim=2)) # input\_dim alterado para 2

model.add(Dense(3, activation='sigmoid'))

sgd = optimizers.SGD(learning\_rate=10)

model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])

# Dados de treino

ax = np.matrix('4 6; 15 8; 65 90; 5 90; 18 70; 70 10', dtype="float32")

ax = 2 \* (ax / 100 - 0.5) # -1 a +1

ay = np.matrix('0 1 0; 0 0 1; 1 0 0; 0 1 0; 0 0 1; 1 0 0', dtype="float32")

model.fit(ax, ay, epochs=200, batch\_size=2, verbose=2)

# Dados de teste

qx = np.matrix('20 6; 3 50; 75 55; 15 8; 70 10', dtype="float32")

qx = 2 \* (qx / 100 - 0.5) # -1 a +1

qy = np.matrix('0 0 1; 0 1 0; 1 0 0; 0 0 1; 1 0 0', dtype="float32")

teste = model.evaluate(qx, qy)

print(f"Custo e acuracidade de teste: {teste}")

# Classificação de teste

qp2 = model.predict(qx)

print(f"Classificação de teste:\n {qp2}")

qp = qp2.argmax(axis=1)

print(f"Rótulo de saída:\n {qp}")

# Plot do modelo e sumário

keras.utils.plot\_model(model, to\_file='abc1.png', show\_shapes=True)

model.summary()

**Explicação:**

-

**Código da lição:**

| // Lição de casa 5 (Parte 2)  //  // Gustavo Henrique da Silva Amaral - 12551686  // Thiago da Rocha Calomino Gonçalves - 12554647 |
| --- |