[PSI3472 aula 10, Início]

I. Meta-Learning

Vamos usar o exemplo do vídeo [https://www.youtube.com/watch?v=hE7eGew4eeg] para ilustrar meta-learning. Suponha que alguém mostre a você a figura F1, com 2 imagens de tatus e 2 de pangolins. Depois, mostre a imagem de busca da figura F2 e peça para você classificá-la. Você não terá dificuldade em classificar o animal da figura F2 como pangolim, mesmo que não conheça esses animais. Por outro lado, uma rede neural necessita de milhares de imagens de tatus e outras tanto de pangolins para aprender a distinguir as duas classes.

(a) Tatu

(b) Pangolim

(c) Tatu

(d) Pangolim

Figura F1: Conjunto suporte, com dois tatus e dois pangolins.

Figura F2: Imagem de busca (query).

Por que você consegue distinguir facilmente tatus dos pangolins usando apenas dois exemplos de cada animal, enquanto que uma rede neural necessita de milhares de exemplos? É que você já viu muitos outros animais, plantas e objetos e aprendeu a extrair os atributos da imagem que caracterizam cada ser. No caso de tatus e pangolins, os dois animais têm escamas e orelhas diferentes. Se a rede neural soubesse como extrair os principais atributos que caracterizam cada animal, ela saberia

identificar facilmente novas espécies (para as quais ainda não foi treinada) a partir de poucas imagens de treino.

Existe uma área de pesquisa denominada de *meta learning*, que consiste em fazer uma rede neural "aprender a aprender". Já vimos uma destas técnicas: *transfer learning*: estudamos como transfer learning facilita distinguir rostos masculinos dos femininos, usando um modelo treinado originalmente para reconhecer as 1000 classes da ImageNet, sem ter sido treinado para reconhecer rostos humanos.

A área de meta learning é vasta. Vamos estudar somente algumas técnicas simples, entre elas one-shot learning, reconhecimento facial e few-shot learning.

II. Few/One-Shot Learning

Vamos seguir as explicações dos posts [Brownlee2019, Lamba2019, Bouma2017] para aprender sobre few/one-shot learning. Também vamos seguir as explicações de:

https://blog.paperspace.com/few-shot-learning
https://www.youtube.com/watch?v=hE7eGew4eeg
https://www.youtube.com/watch?v=4S-XDefSjTM
https://www.youtube.com/watch?v=U6uFOIURcD0

Já vimos que para fazer classificação com CNN é necessária uma grande quantidade de dados de treino. Por exemplo, para ensinar CNN a decidir se numa imagem aparece uma pessoa *P* ou não, são necessárias muitas fotos de *P* juntamente com muitas fotos de pessoas diferentes de *P*. Few/one-shot learning permite efetuar classificação a partir de poucas ou mesmo uma única imagem de *P*. Isto parece mágica, mas veremos que na verdade aprendizagem few/one-shot utilizou previamente uma grande quantidade de imagens de rostos humanos para fazer um "pré-treino" para reconhecer se duas fotos se referem à mesma pessoa ou não, de forma que mais tarde necessita de uma única imagem de treino por classe.

Considere uma empresa que quer instalar um sistema para fazer reconhecimento facial de seus funcionários. Para treinar um classificador CNN, são necessárias dezenas ou centenas de imagens de cada funcionário fotografado sob diferentes ângulos, iluminações, expressões faciais, penteados, etc. Porém, é complicado para empresa conseguir essa quantidade de fotos de cada funcionário. Além disso, mesmo que consiga muitas fotos de cada funcionário, se um funcionário está sem barba em todas as fotos-exemplos o sistema poderá não reconhecê-lo se um dia aparecer de barba. Os problemas não terminam aí. Digamos que a empresa tinha 10 funcionários e treinou o sistema de reconhecimento de face fornecendo centenas fotografias de cada funcionário. O treino demorou várias horas de processamento. Aí a empresa contratou mais um funcionário. A empresa vai ter que refazer todo o treino novamente.

Few/one-shot learning aprende, a partir de uma quantidade grande de pares de fotos de pessoas classificadas como "iguais" ou "diferentes", o que faz duas fotos serem da mesma pessoa. Isto é, aprende uma "função distância" que devolve um valor baixo se as duas fotos forem da mesma pessoa e devolve um valor alto se representarem duas pessoas diferentes. Este treino é feito previamente pelo fabricante do sistema de reconhecimento facial, longe do usuário final do sistema (há vários modelos gratuitos pré-treinados por grandes empresas). Depois, basta inserir uma única foto (ou poucas fotos) de cada funcionário/condômino para verificar se a pessoa que está sendo fotografada pela câmera de segurança é a mesma pessoa (ou não) daquela armazenada no conjunto de dados. Se a "função distância" tiver sido bem projetada, ela irá reconhecer a face independentemente de usar óculos ou não, de estar com barba ou não, de usar penteados diferentes, etc.

Se a empresa contratar um novo funcionário, basta adicionar uma única foto (ou poucas fotos) dele no BD que o sistema continuará funcionando sem a precisar repetir o treino. Se a empresa demitir um funcionário, bastará remover a foto do funcionário demitido do BD.

Outro exemplo de uso de "one-shot learning" é a verificação de assinaturas. Basta ter poucas assinaturas de cada cliente para que o sistema possa verificar se uma dada assinatura é do cliente registrado (ou não). [Jade2022]

Para facilitar a descrição de alguns conceitos envolvidos, vamos considerar o exemplo de distinguir tatu do pangolim.

Conjunto de treino: Consiste de milhões de fotos de animais, com rótulo que identifica o animal. Este conjunto será usado para pré-treinar o modelo de few/one-shot learning.

Conjunto de suporte: Consiste de uma ou poucas fotos de cada animal novo que queremos que o sistema reconheça. Por exemplo, duas fotos de tatu e duas de pangolim.

Conjunto de consulta: Consiste em imagens que o modelo irá classificar em uma das classes do conjunto de suporte. Por exemplo, uma foto de pangolim.

Esquema de aprendizado de N-way K-shot: Esta é uma frase comum usada na literatura few-shot learning. "*N*-way" indica que há *N* novas categorias nas quais um modelo pré-treinado precisa generalizar (ex: *N*=2 classes de animais desconhecidos). "*K*-shot" define o número de amostras rotuladas no conjunto de suporte para cada uma das *N* novas classes. Por exemplo, *K*=2 indica que há duas fotos de tatu e duas de pangolim. Assim, o problema do tatu/pangolim seria esquema de aprendizado 2-way 2-shot.

Nota: Um sistema que reconhece 100 moradores de um condomínio a partir de 1 foto de cada pessoa não se encaixa perfeitamente como problema 100-way 1-shot, pois uma pessoa na portaria pode não ser morador.

III. Reconhecimento de Caractere

1. Banco de dados Omniglot

Seguindo [Lamba2019, Bouma2017, Lake2019], vamos usar o conjunto de imagens Omniglot para testar one-shot learning. Este conjunto é uma coleção de 1623 caracteres de 50 diferente alfabetos. Há 20 exemplos escritos por diferentes pessoas para cada um dos 1632 caracteres, na resolução 105×105, resultando em 20×1632=32640 imagens. Há 964 caracteres para o treino e 659 caracteres para o teste.

O problema é treinar com os dados de treino uma rede que diz se duas imagens representam um mesmo caractere (ou não). Depois, dadas duas imagens do conjunto de teste, CNN deve responder se elas representam um mesmo caractere (ou não).

Nota: Este problema seria "análogo" a treinar reconhecimento de face usando fotografias de 964 pessoas diferentes, com 20 fotos de cada pessoa. Depois, testar o reconhecimento de face usando fotografias de 659 outras pessoas.

Você pode baixar Omniglot de [Lake2019], copiando images_background.zip (imagens de treino) e images_evaluation.zip (imagens de teste). Ou pode usar o código do Programa P, que baixa e descompacta esse conjunto no seu diretório atual (no Colab ou no computador local):

```
import os;
lista=['https://github.com/brendenlake/omniglot/raw/master/python/images_background.zip', \
     'https://github.com/brendenlake/omniglot/raw/master/python/images_evaluation.zip']
for url in lista:
  nomeArq=os.path.split(url)[1]
  if not os.path.exists(nomeArq):
    print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
    os.system("wget -nc -U 'Firefox/50.0' "+url)
  else:
    print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
  print("Descompactando arquivos novos de",nomeArq)
  os.system("unzip -u "+nomeArq)
```

Programa P: Faz download do Omniglot.
https://colab.research.google.com/drive/1MbqZFFMtuhKJWgFmAVUyBgUWbqIGMFfP

A figura 1 mostra alguns caracteres de Omniglot. A figura 2 mostra as 20 imagens de um único caractere de Omniglot.

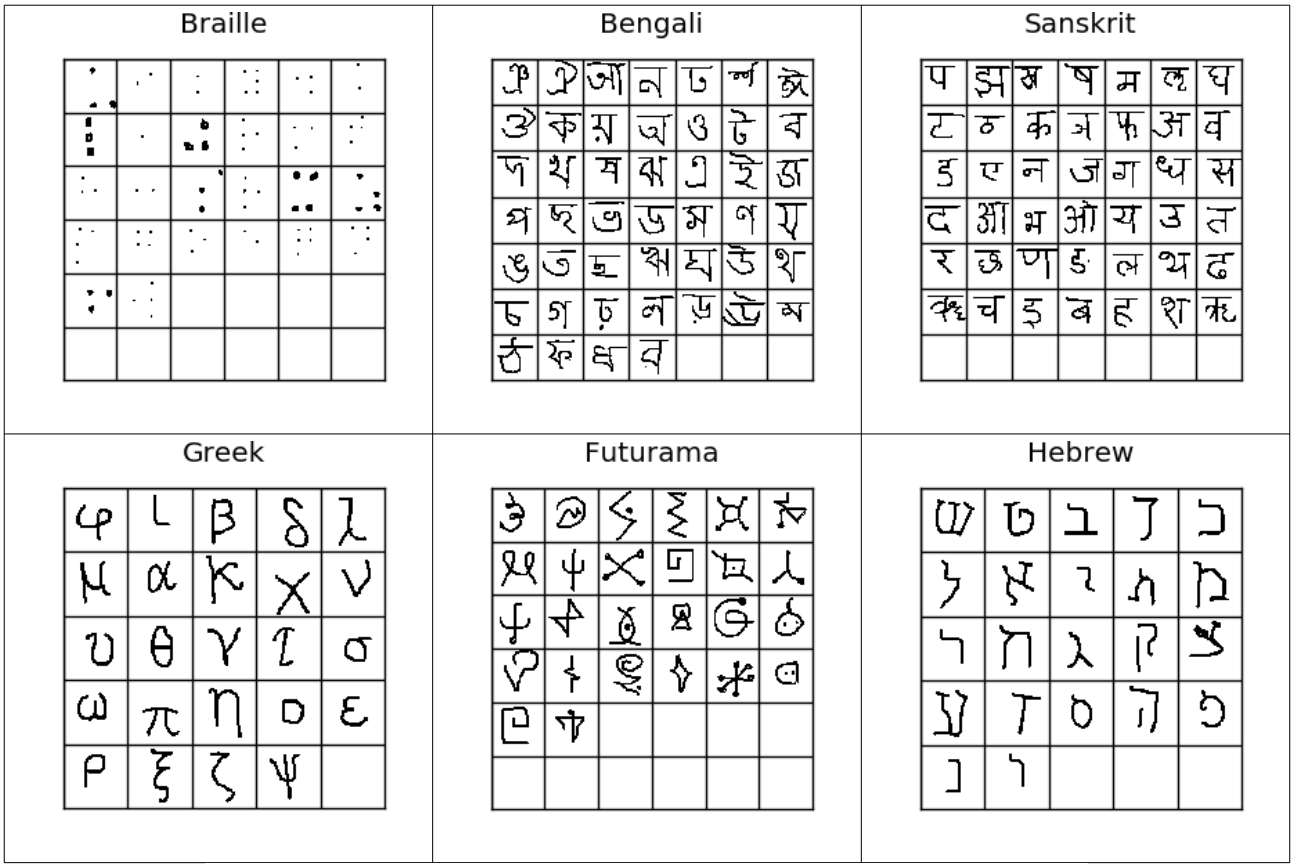


Figura 1: Alguns caracteres de Omniglot. Imagem de [Bouma2017].

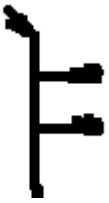
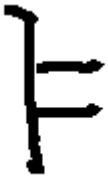
 0645_01.png	 0645_02.png			 0645_05.png
 0645_06.png				
 0645_11.png				
 0645_16.png				 0645_20.png

Figura 2: Exemplo de 20 imagens que representam um mesmo caractere.

Após baixar e descompactar Omniglot, haverá dois subdiretórios no seu diretório atual: images_background (imagens de treino) e images_evaluation (imagens de teste), cada um com mais subdiretórios.

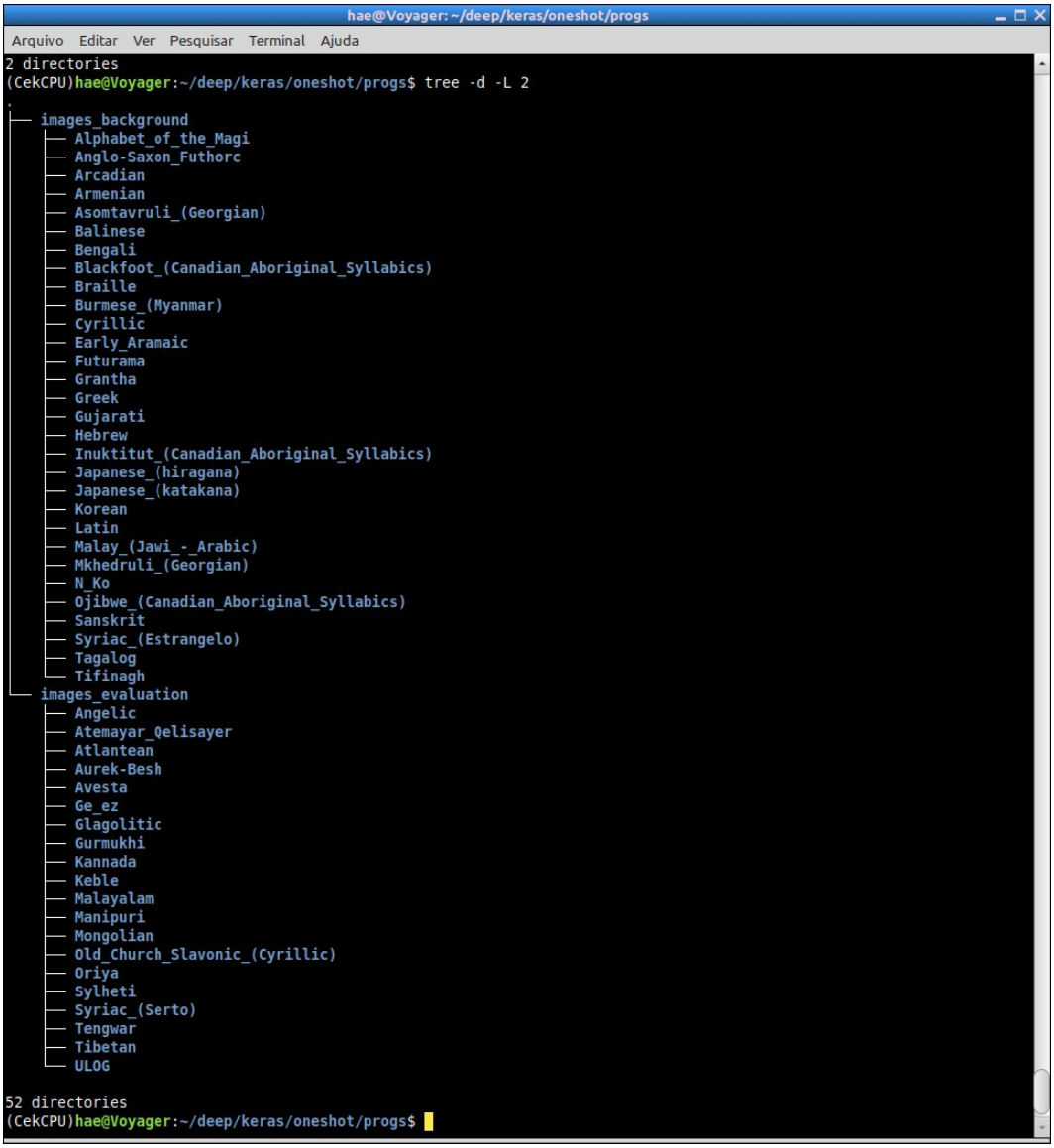


Figura 3: Estrutura do conjunto Omniglot.

2. Rede siamesa para one-shot learning

Rede siamesa para one-shot learning foi proposta por [Koch et al., 2015].
https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf
Vamos usar o blog e a implementação abaixo, fazendo adaptações para simplificar o programa e aumentar a velocidade de treino:
https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d
https://github.com/hlamba28/One-Shot-Learning-with-Siamese-Networks

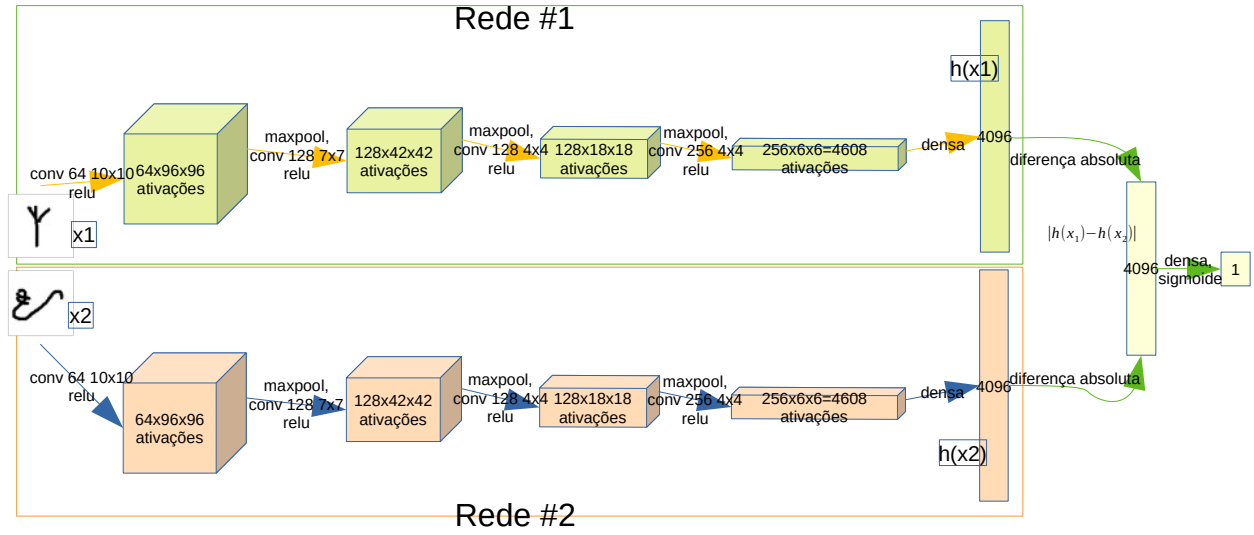


Figura 4: Rede siamesa original, consistindo de duas redes completamente iguais.

O termo “siamesa” significa gêmeos que nascem ligados por uma parte do corpo. A rede siamesa consiste de duas redes convolucionais completamente iguais: as duas redes possuem estruturas e todos os parâmetros (pesos e vieses) iguais (figura 4). Na verdade, não temos duas redes mas uma única rede que recebe duas imagens diferentes e gera duas saídas também diferentes. Assim, talvez a figura 5 seja uma representação melhor.

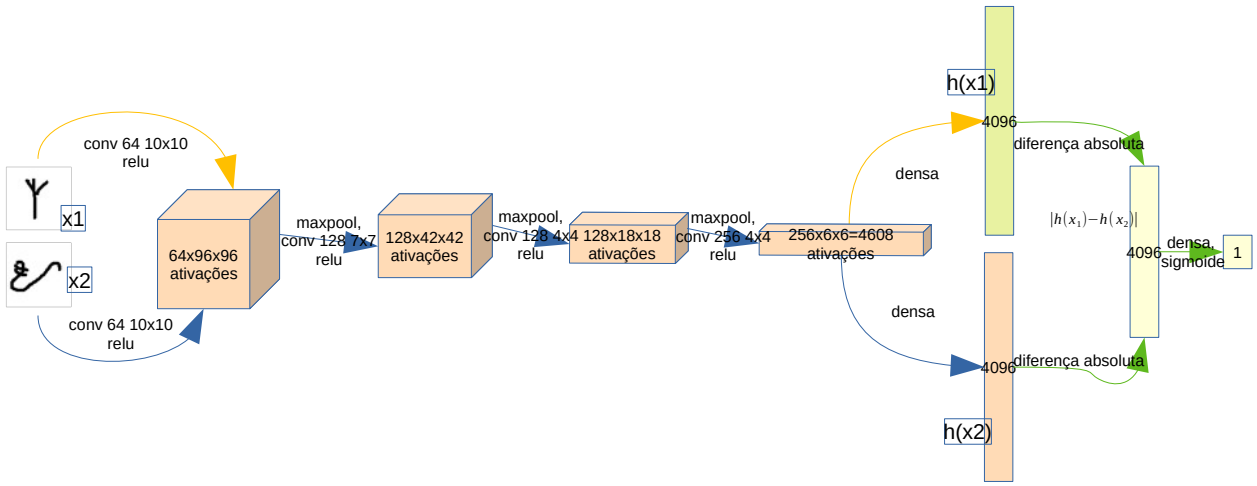


Figura 5: Na verdade, uma rede siamesa consiste de uma única rede que processa duas imagens X_1 e X_2 diferentes, gerando dois vetores de atributos $h(X_1)$ e $h(X_2)$.

A solução original usava imagens de Omniglot com resolução original excessivamente grande (105×105). Para acelerar o processamento, vamos redimensionar as imagens para 64×64. Além disso, a rede usava convoluções grandes (10×10, 7×7, etc.). Já vimos que é possível substituir convoluções grandes por uma sequência de convoluções 3×3, melhorando tanto a acuracidade como a velocidade de processamento. Fazendo isso, resulta numa estrutura de rede "inspirada" em VGG (figura F1).

Na figura F1, as duas imagens x_1 e x_2 passam pela mesma rede *seq*, gerando dois vetores de 1600 atributos $h(x_1)$ e $h(x_2)$. A estrutura interna da rede *seq* está representada nas figuras F2(a) e F3(a). A rede *seq* é inserida dentro de uma segunda rede *siamese_net* (figuras F2(b) e F3(b)) que calcula a diferença absoluta $|h(x_1)-h(x_2)|$, gerando um vetor de 1600 atributos. Esse vetor passa por uma camada densa seguida de sigmoide para resultar no valor de saída entre 0 e 1. Queremos que a saída seja próxima de "0" se as duas imagens de entrada representarem caracteres iguais, e que seja próxima de "1" caso contrário. O programa P é a implementação em Keras.

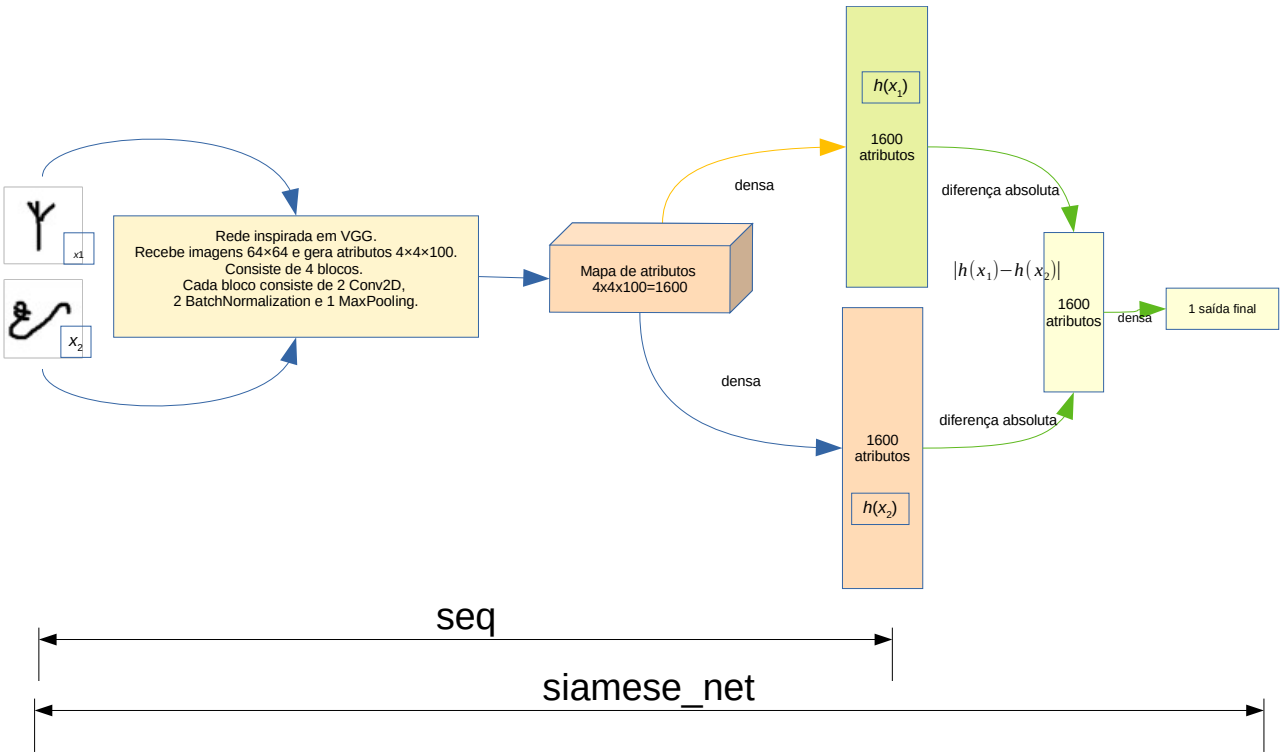
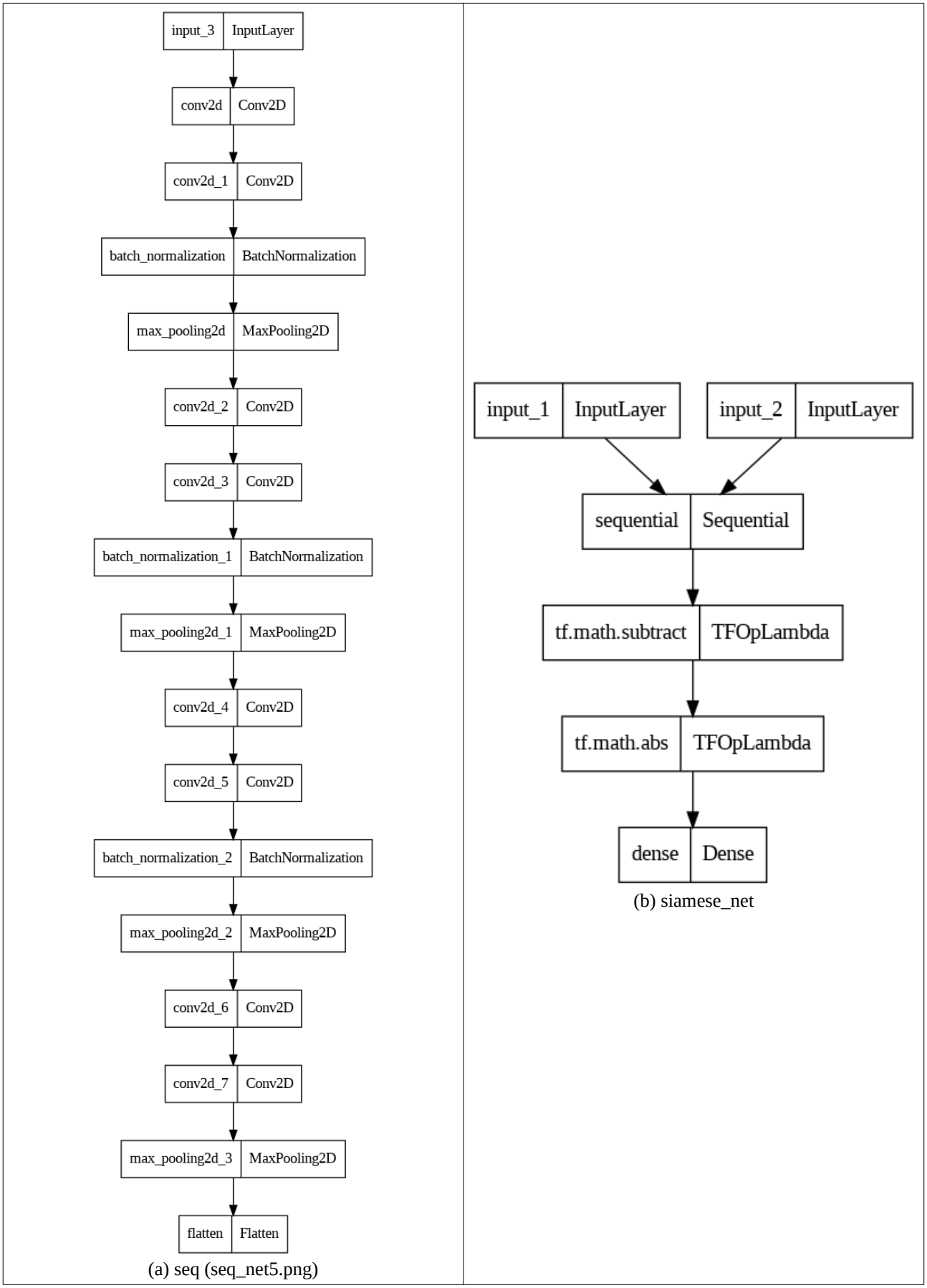


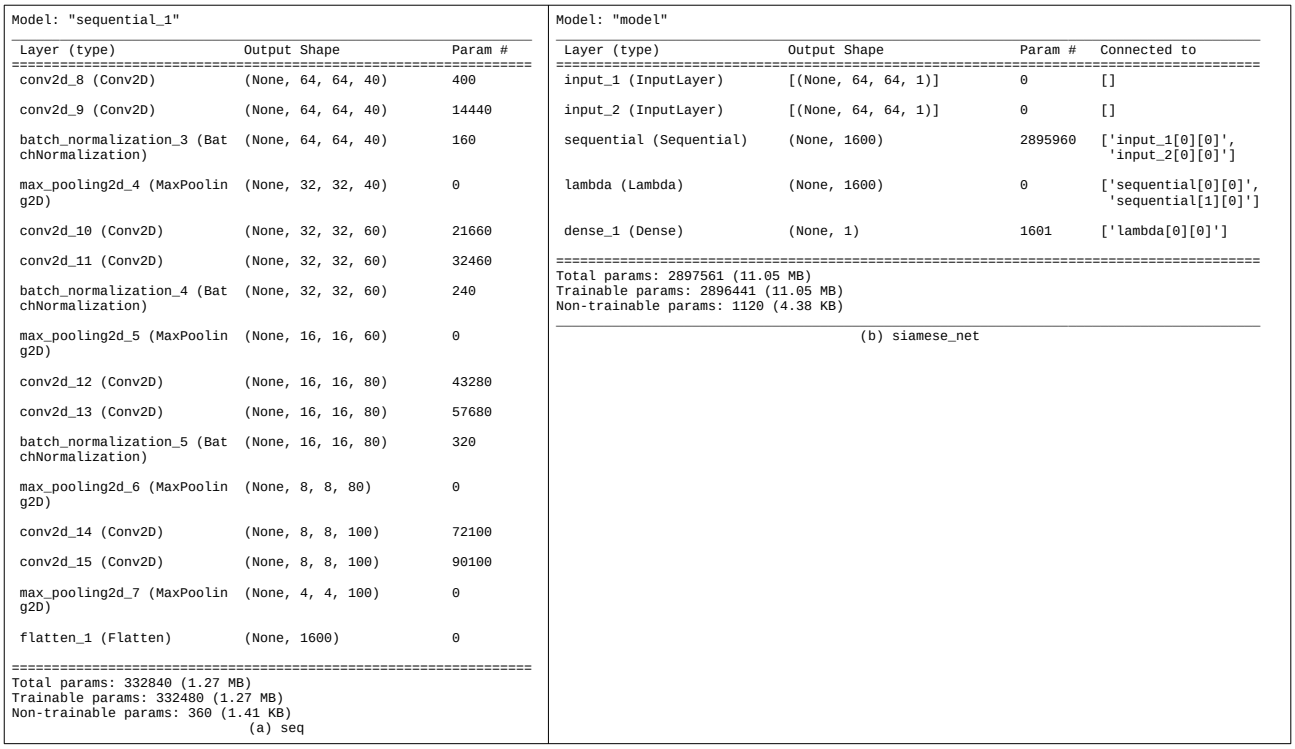
Figura F1: Rede siamesa usando rede convolucional semelhante à VGG. Parte compartilhada da rede é *seq*. A rede completa é *siamese_net*.



(a) seq (seq_net5.png)

(b) siamese_net

Figura F2: (a) Rede *seq* "inspirada" em VGG que recebe duas imagens e gera dois conjuntos de 1600 atributos. (b) Rede *siamese_net* que recebe dois conjuntos de 1600 atributos e decide se eles se referem (ou não) a um mesmo caractere.



Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 64, 64, 40)	400
conv2d_9 (Conv2D)	(None, 64, 64, 40)	14440
batch_normalization_3 (BatchNormalization)	(None, 64, 64, 40)	160
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 40)	0
conv2d_10 (Conv2D)	(None, 32, 32, 60)	21660
conv2d_11 (Conv2D)	(None, 32, 32, 60)	32460
batch_normalization_4 (BatchNormalization)	(None, 32, 32, 60)	240
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 60)	0
conv2d_12 (Conv2D)	(None, 16, 16, 80)	43280
conv2d_13 (Conv2D)	(None, 16, 16, 80)	57680
batch_normalization_5 (BatchNormalization)	(None, 16, 16, 80)	320
max_pooling2d_6 (MaxPooling2D)	(None, 8, 8, 80)	0
conv2d_14 (Conv2D)	(None, 8, 8, 100)	72100
conv2d_15 (Conv2D)	(None, 8, 8, 100)	90100
max_pooling2d_7 (MaxPooling2D)	(None, 4, 4, 100)	0
flatten_1 (Flatten)	(None, 1600)	0

Total params: 332840 (1.27 MB)
Trainable params: 332480 (1.27 MB)
Non-trainable params: 360 (1.41 KB)

(a) seq

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 64, 64, 1)]	0	[]
input_2 (InputLayer)	[(None, 64, 64, 1)]	0	[]
sequential (Sequential)	(None, 1600)	2895960	['input_1[0][0]', 'input_2[0][0]']
lambda (Lambda)	(None, 1600)	0	['sequential[0][0]', 'sequential[1][0]']
dense_1 (Dense)	(None, 1)	1601	['lambda[0][0]']

Total params: 2897561 (11.05 MB)
Trainable params: 2896441 (11.05 MB)
Non-trainable params: 1120 (4.38 KB)

(b) siamese_net

Figura F3: (a) Rede *seq* “inspirada” em VGG que recebe duas imagens e gera dois conjuntos de 1600 atributos. (b) Rede *siamese_net* que recebe dois conjuntos de 1600 atributos e decide se eles se referem (ou não) a um mesmo caractere.

```
# -*- coding: utf-8 -*-
"""oneshot5.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1n_xyYcWuO11_z5UjKK9ApXmeE7RzAS7g

Oneshot5.py: Faz oneshot learning para classificar Omniglot
"""

#No Colab, trocar Tensorflow 2.17 com bug por Tensorflow 2.15
!pip uninstall tensorflow -y --quiet
!pip install tensorflow==2.15 --quiet

train_folder = "./images_background/"
val_folder = './images_evaluation/'
save_path = "."
model_path = "."
nl=64; nc=64 # Imagem original 105x105. Sera redimensionado para 64x64.

#Baixa Omniglot de: https://github.com/brendenlake/omniglot
import os;
lista=['https://github.com/brendenlake/omniglot/raw/master/python/images_background.zip', \
     'https://github.com/brendenlake/omniglot/raw/master/python/images_evaluation.zip']
for url in lista:
  nomeArq=os.path.split(url)[1]
  if not os.path.exists(nomeArq):
    print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
    os.system("wget -nc -U 'Firefox/50.0' "+url)
  else:
    print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
  print("Descompactando arquivos novos de",nomeArq)
  os.system("unzip -u "+nomeArq)

# Abaixo, uma adaptação do programa: https://github.com/hlamba28/One-Shot-Learning-with-Siamese-Networks
# O blog original está em: https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d
import sys
import numpy as np
import pickle, os, cv2, time
import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate
from tensorflow.keras.utils import load_img
import tensorflow.keras.initializers as initializers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization, MaxPooling2D, Concatenate, GlobalAveragePooling1D
from tensorflow.keras.layers import Layer, Lambda, Flatten, Dense, Input
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.regularizers import l2
from tensorflow.keras import backend as K
from sklearn.utils import shuffle
from tensorflow.keras.utils import plot_model
import numpy.random as rng

# Funcao para carregar imagens nl*nc, com pixels entre -0.5 e +0.5
def loadimgs(path, target_size, n = 0):
    #path => Path of train directory or test directory
    X=[]; y=[]
    cat_dict = {}; lang_dict = {}; curr_y = n
    # we load every alphabet seperately so we can isolate them later
    for alphabet in os.listdir(path):
        #print("loading alphabet: " + alphabet)
        lang_dict[alphabet] = [curr_y,None]
        alphabet_path = os.path.join(path,alphabet)
        for letter in os.listdir(alphabet_path):
            cat_dict[curr_y] = (alphabet, letter)
            category_images=[]
            letter_path = os.path.join(alphabet_path, letter)
            # read all the images in the current category
            for filename in os.listdir(letter_path):
                image_path = os.path.join(letter_path, filename)
                image = load_img(image_path, color_mode="grayscale",
                                 target_size=target_size, interpolation="nearest")
                category_images.append(image)
                y.append(curr_y)
            try:
                X.append(np.stack(category_images))
            # edge case  - last one
            except ValueError as e:
                print(e); print("error - category_images:", category_images)
            curr_y += 1
            lang_dict[alphabet][1] = curr_y - 1
    y = np.vstack(y); X = np.stack(X)
    X = X / 255.0 - 0.5 #Entre -0.5 e +0.5
    return X,y,lang_dict

# Carrega as imagens de treino. Train_classes indica o alfabeto e pode ser ignorado.
# Formato de Xtrain: (964, 20, 64, 64)
# Formato do ytrain (19280, 1)
Xtrain,ytrain,train_classes=loadimgs(train_folder,(nl,nc))
# Carrega as imagens de teste. Val_classes indica o alfabeto e pode ser ignorado.
Xval,yval,val_classes=loadimgs(val_folder,(nl,nc))
# Verifica o formato dos dados lidos
# print("Formato de Xtrain:",Xtrain.shape)
# print("Uma linha do primeiro caractere:",Xtrain[0,0,30])
# print("Formato do ytrain",ytrain.shape)
# print("Alguns rotulos de ytrain",ytrain[0:3],ytrain[-3:-1])
# print("train_classes",train_classes)
# print("Formato de Xval:",Xval.shape)
```

```
105 
106 # Funcao que pega uma lote com batch_size de pares de imagens
107 # Metade dos pares de classes diferentes e outra metade dos pares de classes iguais
108 def get_batch(batch_size, s="train"):
109     """Create batch of n pairs, half same class, half different class"""
110     if s == 'train':
111         X = Xtrain; categories = train_classes; replace=False
112     else:
113         X = Xval; categories = val_classes; replace=True
114     n_classes, n_examples, w, h = X.shape
115 
116     categories = rng.choice(n_classes, size=(batch_size,), replace=replace)
117     pairs=[np.zeros((batch_size, h, w, 1)) for i in range(2)]
118     targets=np.ones((batch_size,))
119     targets[batch_size//2:] = 0
120     for i in range(batch_size):
121         category = categories[i]
122         idx_1 = rng.randint(0, n_examples)
123         pairs[0][i,:,:,:] = X[category, idx_1].reshape(w, h, 1)
124         idx_2 = rng.randint(0, n_examples)
125         # pick images of same class for 1st half, different for 2nd
126         if i >= batch_size // 2:
127             category_2 = category
128         else:
129             # add a random number to the category modulo n classes to ensure 2nd image has a different category
130             category_2 = (category + rng.randint(1,n_classes)) % n_classes
131         pairs[1][i,:,:,:] = X[category_2,idx_2].reshape(w, h,1)
132     return pairs, targets
133 
134 #Imprime batch com 4 pares
135 def print_batch4(pairs, targets):
136   f = plt.figure(figsize=(2,4))
137   for i in range(4):
138     f.add_subplot(4,2,2*i+1)
139     plt.imshow(np.squeeze(pairs[0][i]), cmap="gray")
140     plt.axis("off");
141     f.add_subplot(4,2,2*i+2)
142     plt.imshow(np.squeeze(pairs[1][i]), cmap="gray")
143     plt.axis("off");
144     plt.text(0, 2, int(targets[i]), color="r")
145   plt.savefig('oneshot5_teste_pares.png')
146   plt.show()
147 
148 # Escolhe aleatoriamente lote de 4 pares de imagens e imprime
149 # 2 pares da mesma classe e 2 de classes diferentes
150 pairs, targets = get_batch(4)
151 print_batch4(pairs, targets)
152 
153 # Funcao que constroi a rede siamesa
154 def get_siamese_model(input_shape):
155     # Define the tensors for the two input images
156     left_input = Input(input_shape); right_input = Input(input_shape)
157 
158     # Convolutional Neural Network
159     seq = Sequential() #64x64x1
160     seq.add(Input(input_shape))
161 
162     seq.add(Conv2D(40, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
163     seq.add(Conv2D(40, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
164     seq.add(BatchNormalization())
165     seq.add(MaxPooling2D(pool_size=(2,2))) #32x32x40
166 
167     seq.add(Conv2D(60, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
168     seq.add(Conv2D(60, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
169     seq.add(BatchNormalization())
170     seq.add(MaxPooling2D(pool_size=(2,2))) #16x16x60
171 
172     seq.add(Conv2D(80, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
173     seq.add(Conv2D(80, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
174     seq.add(BatchNormalization())
175     seq.add(MaxPooling2D(pool_size=(2,2))) #8x8x80
176 
177     seq.add(Conv2D(100, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
178     seq.add(Conv2D(100, kernel_size=(3,3), activation='relu', padding='same', kernel_regularizer=l2(2e-4)))
179     seq.add(MaxPooling2D(pool_size=(2,2))) #4x4x100
180 
181     seq.add(Flatten())
182     #seq.add(Dense(1600, activation='sigmoid', kernel_regularizer=l2(1e-4) ))
183 
184     # Generate the encodings (feature vectors) for the two images
185     encoded_l = seq(left_input); encoded_r = seq(right_input)
186 
187     # L1_layer = Lambda(lambda tensors:tf.math.abs(tensors[0] - tensors[1]))
188     def L1_layer(a,b):
189       return tf.math.abs(a-b)
190     L1_distance = L1_layer(encoded_l, encoded_r)
191     # L1_distance= BatchNormalization()(L1_distance)
192 
193     prediction = Dense(1,activation='sigmoid')(L1_distance)
194     # O engracado e' que se eu troco a camada densa pela média como abaixo, o treino nao converge
195     # def reduce_mean(a):
196     #   return tf.math.reduce_mean(a,1)
197     # prediction = reduce_mean(L1_distance)
198     # prediction = keras.layers.Activation("sigmoid")(prediction)
199 
200     # Connect the inputs with the outputs
201     siamese_net = Model(inputs=[left_input,right_input], outputs=prediction)
202 
203     # return the model
204     return siamese_net, seq
205 
206 # Execute esta celula para construir a rede "do zero" e imprimir a arquitetura
207 model, seq = get_siamese_model((nl,nc,1))
208 model.summary(); seq.summary()
209 plot_model(model, to_file=os.path.join(save_path,"siamese_net5.png"))
210 plot_model(seq, to_file=os.path.join(save_path,"seq_net5.png"))
```

```
211
212 optimizer = Adam()
213 model.compile(loss="binary_crossentropy",optimizer=optimizer,metrics=['accuracy'])
214
215 # Faz o treino ou continua o treino de onde parou.
216 batch_size = 32
217 n_iter = 20000 # No. of training iterations. Original=20000
218
219 print("Starting training process!")
220 print("-------------------------------------")
221 t_start = time.time()
222 for i in range(n_iter):
223     (inputs,targets) = get_batch(batch_size)
224     loss = model.train_on_batch(inputs, targets)
225     if i%200==0:
226       print("Epoca=%5d "%(i),"Loss=%.3f Accuracy=%.3f lr=%.3e"%(loss[0],loss[1],model.optimizer.lr))
227       model.optimizer.lr.assign( model.optimizer.lr * 0.98 )
228 model.save(os.path.join(model_path, 'oneshot5.keras'))
229
230 soma_accuracy=0
231 for i in range(10):
232   # Gera um lote de teste e verifica a taxa de acerto
233   qx,qy=get_batch(10000,"test")
234   #qp = model.predict(qx)
235   results=model.evaluate(qx,qy)
236   soma_accuracy += results[1]
237 print("Taxa de acerto media:",soma_accuracy/10)
238
239 # Verifica se os atributos de classes iguais sao semelhantes
240 # np.set_printoptions(precision=3)
241 # n=4
242 # qx,qy=get_batch(n,s="test")
243
244 # print("qx[0].shape",qx[0].shape)
245 # print("qx[1].shape",qx[1].shape)
246 # print("qy.shape",qy.shape)
247 # qp0=seq.predict(qx[0])
248 # qp1=seq.predict(qx[1])
249 # qp=model.predict(qx)
250 # print("qp0.shape",qp0.shape)
251 # print("qp1.shape",qp1.shape)
252 # print("qp.shape",qp.shape)
253 # for i in range(n):
254 #   print("qp0[%d,0:10]"%(i),qp0[i,0:10])
255 #   print("qp1[%d,0:10]"%(i),qp1[i,0:10])
256 #   print("qp[%d,0:10]"%(i),qp[i,0:10])
257 # print_batch4(qx,qy)
```

Programa P: Programa oneshot5.py que faz one shot learning.
https://colab.research.google.com/drive/1n_xyYcWuO11_z5UjKK9ApXmeE7RzAS7g

1) Linhas 52-89: A função *loadimgs* carrega o conjunto de imagens, juntamente com os rótulos (números de 0 a 1623). O formato deles é:

Formato de *Xtrain*: (964, 20, 64, 64)
Formato de *ytrain*: (19280, 1)
Formato de *Xval*: (659, 20, 64, 64)
Formato de *yval*: (13180, 1)

Indicando que tensor *Xtrain* possui 964 entradas (caracteres) e cada entrada com 20 imagens 105×105. Os valores dos pixels das imagens são -0.5 (preto) ou +0.5 (branco). O vetor *ytrain* possui 19280=964×20 entradas que indicam o caractere correspondente a cada imagem, números de 0 a 963. Como há 20 imagens para cada caractere, as entradas 0 a 19 são de caractere 0, as entradas de 20 a 39 são de caractere 1, etc.

Nota: O dicionário *train_classes* indica o alfabeto (Armenian, Balinese, Katakana, etc) ao qual o caractere pertence. Não usaremos este dado no nosso programa.

2) Linhas 101-143: A função *get_batch* gera aleatoriamente *batch_size* pares de imagens, metade delas de classes diferentes (distância “1”) e outra metade de classes iguais (distância “0”). Por exemplo:

```
pairs,targets=get_batch(4)
```

gera 4 pares de imagens, metade com rótulo “1” e outra metade com rótulo “0”. Vamos alimentar a rede siamesa com lotes de pares de imagens como na figura 9.

3) Linhas 126-199: A função *get_siamese_model* constroi e devolve a rede *siamese_net*. Além disso, devolve também a rede *seq*, a parte da rede *siamese_net* que é compartilhada pelas duas imagens. Depois de treinada, é possível usar estas redes de duas formas:

a) Dadas duas imagens de dois caracteres, calcular a diferença entre elas usado a rede *siamese_net*.

b) Dada a imagem de um caractere, calcular os seus atributos usando a sub-rede *seq*.

Figura 9: Alguns lotes com classes diferentes “1” e iguais “0”

4) Linhas 203-221: Treina o modelo. Executando, obtemos saída como:

```
Epoca=    0  Loss=0.892 Accuracy=0.500 lr=1.000e-03
Epoca= 2000  Loss=0.295 Accuracy=0.969 lr=8.171e-04
Epoca= 4000  Loss=0.173 Accuracy=0.969 lr=6.676e-04
Epoca= 6000  Loss=0.123 Accuracy=0.969 lr=5.455e-04
Epoca= 8000  Loss=0.114 Accuracy=1.000 lr=4.457e-04
Epoca=10000  Loss=0.261 Accuracy=0.969 lr=3.642e-04
Epoca=12000  Loss=0.099 Accuracy=1.000 lr=2.976e-04
Epoca=14000  Loss=0.314 Accuracy=0.969 lr=2.431e-04
Epoca=16000  Loss=0.074 Accuracy=1.000 lr=1.986e-04
Epoca=18000  Loss=0.100 Accuracy=0.969 lr=1.623e-04
Epoca=19800  Loss=0.057 Accuracy=1.000 lr=1.353e-04
```

Repare que loss e accuracy ficam diminuindo e aumentando, provavelmente pelo fato de não usar uma função de perda inadequada. Talvez isto melhore se usar triplet loss explicado adiante.

5) Linhas 223-230: Depois de treinar a rede, temos um modelo *siamese_net* que recebe duas imagens e retorna uma “nota” entre 0 e 1 indicando se as duas imagens representam (ou não) o mesmo caractere. Para testar o modelo, vamos gerar aleatoriamente *n* pares de imagens, metade representando caracteres iguais e outra metade caracteres diferentes. Depois, vamos alimentar a rede siamese com esses pares de imagens. Consideraremos que, se a resposta da rede for menor que 0,5, a rede classificou o par como caracteres iguais. Fazendo isso com 10×10000 pares de imagens, obtive taxa média de acerto 95,41%. Nota: Isto é consideravelmente melhor que o programa original do [blog], que estava acertando 89,41%.

Exercício: Em vez de limiarizar em 0,5 como acima, trace a curva ROC e calcule AUC e a taxa de acerto no ponto EER (equal error rate).

Exercício: Use data augmentation para aumentar a taxa de acerto.

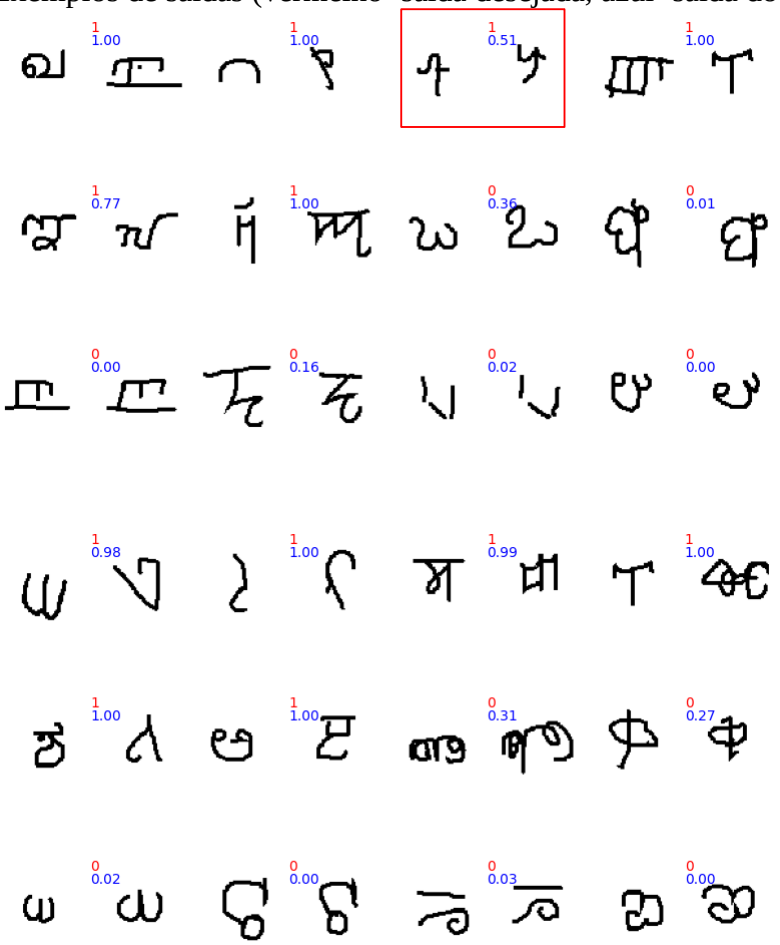
Exercício: Escreva um programa que:
a) Carrega o modelo *oneshot5.keras* já treinado com o programa P (oneshot5.py):
https://drive.google.com/file/d/1LrJz-IhAixFnFtcaFh3H8v3G69Dhoodn

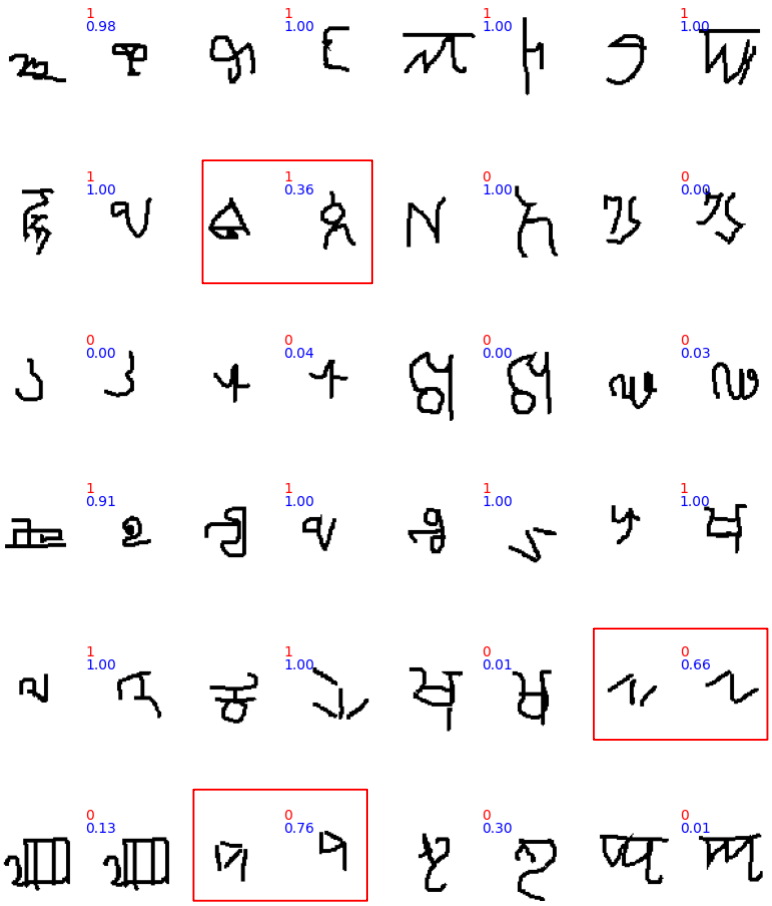
```
# Execute esta celula para carregar a rede ja treinada
import os; import gdown
id="1LrJz-IhAixFnFtcaFh3H8v3G69Dhoodn"
nomeArq="oneshot5.keras"
if not os.path.exists(nomeArq):
  print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
  gdown.download(id=id, output=nomeArq)
else:
  print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
```

b) Lê nomes de duas imagens de OmniGlot e mostra as duas imagens na tela, juntamente com a distância (a resposta da rede) entre elas.
c) Teste o seu programa fornecendo 6 pares de imagens de caracteres iguais e 6 pares de imagens de caracteres diferentes. Considerando limiar=0.5, qual foi a taxa de acerto?

Solução privada em: https://colab.research.google.com/drive/1dBZDRvAZLcu6vEAfcc_PQDJpeFzlkYaa

Exemplos de saídas (vermelho=saída desejada, azul=saída do modelo):





Exercício: Troque a última camada densa da rede *siamese_net* pelo calculo da média. A taxa de acerto piora muito?

Nota: Tentativa privada que não deu certo: https://colab.research.google.com/drive/1GEboYFHsr-r5VhTknnvfEg9i5T7qW30p?usp=drive_link

3. Triplet loss

Não é razoável esperar que uma rede neural consiga gerar saída exatamente 0 toda vez que as duas imagens de entrada forem da mesma classe, pois duas fotos de tatus (por exemplo) podem ser muito diferentes. Da mesma forma, não é razoável esperar que a rede neural consiga gerar saída exatamente 1 se as duas imagens são de classes diferentes, pois o par de imagens de tatu e de pangolim (por exemplo) podem ser muito semelhantes. Ao forçar que a rede gere saídas 0 ou 1, podemos na verdade piorar o desempenho da rede, ocasionando sobre-ajuste. O que queremos a rede alimentada por duas imagens de classes iguais produza saída substancialmente menor do quando alimentada por um par de imagens de classes diferentes. Triplet loss implementa essa intuição.

Triplet loss $L(P, A, N)$ calcula a função custo de uma tripla de imagens (em vez de um par): uma imagem âncora A, uma imagem positiva P (da mesma classe que A), e uma imagem negativa N (de classe diferente de A). A rede neural calcula os atributos $h(P)$, $h(A)$ e $h(N)$ e depois calcula os quadrados das diferenças euclidianas entre os atributos:

$$d_P = \|h(P) - h(A)\|_2^2 \quad \text{e} \quad d_N = \|h(A) - h(N)\|_2^2$$

O que queremos é que d_P seja bem menor que d_N.

Exemplo: Na figura G, queremos que a diferença entre os atributos das imagens P e A seja bem pequena, pois pertencem à mesma classe pangolim. Similarmente, queremos que a diferença entre os atributos das imagens A e N seja bem grandes, pois pertencem às classes diferentes (pangolim e tatu).

Isto é, queremos que:

$$d_N \geq d_P + \alpha \quad \text{onde } \alpha > 0 \text{ é uma margem.}$$

Se a desigualdade acima for satisfeita, podemos assumir que a rede está gerando atributos desejáveis e portanto não já perda. Caso contrário, assumimos que a perda é:

$$d_P - d_N + \alpha$$

Isto pode ser expressa pela seguinte função triplet loss:

$$L(P, A, N) = max\left\{d_P - d_N + \alpha,\ 0\right\}$$

Função triplet loss calcula a função perda a partir de 3 exemplos, e não 2 exemplos como temos feito até agora.

[PSI3472 aula 9/10. Lição de casa opcional #1 de 1. Vale 10,0]

Em vez de fazer a lição de casa de modificar FCN, pode fazer este exercício no seu lugar.

Modifique o programa *P* para melhorá-lo usando triplet loss. Você também pode introduzir outras melhorias, como fazer data augmentation, etc. Você precisa obter uma taxa de acerto maior do que a do programa *P* (>95,41%).

 a) *P* (amostra positiva)	Amostra positiva *P* é da mesma classe que a amostra âncora *A*. *h*(*P*) são os atributos de *P*. A diferença positiva é $d_P = \|\|h(P) - h(A)\|\|_2^2$.
 b) *A* (amostra âncora)	As amostras *P* e *N* serão comparadas em relação à amostra âncora *A*. *h(A)* são os atributos de *A*.
 c) *N* (amostra negativa)	Amostra negativa *N* é de classe diferente da amostra âncora *A*. *h*(*N*) são os atributos de *N*. A diferença negativa é $d_N = \|\|h(A) - h(N)\|\|_2^2$.

Figura G: Triplet loss $L(P, A, N) = max\left(d_P - d_N + \alpha,\ 0\right)$ calcula a função custo entre 3 amostras: uma amostra âncora *A*, amostra positiva *P* e amostra negativa *N*.

[PSI3472 aula 10, Fim]

[PSI3472 aula 11, Início]

IV. Reconhecimento facial

1. Introdução

Hoje em dia, estamos bem familiarizados com a identificação de pessoas por meio do reconhecimento facial. Porém, há uns 15 anos atrás, grandes empresas e startups estavam trabalhando arduamente em reconhecimento facial, sem obter boas taxas de acerto.

Por exemplo: os telefones celulares podem ser desbloqueados através de reconhecimento facial; acesso a muitos prédios ou condomínios é feito pelo reconhecimento facial; embarque aéreo por reconhecimento facial está disponível na ponte aérea entre Congonhas (SP) e Santos Dumont (RJ), dispensando a apresentação de documentos de identidade (31/10/2022)
[https://www.gov.br/pt-br/noticias/transito-e-transportes/2022/08/entrou-em-funcionamento-a-1a-ponte-aerea-biometrica-do-mundo-para-embarque-de-passageiros].

Tipicamente, o reconhecimento facial compreende 4 etapas:

- Etapa 1: Localizar a face dentro da imagem.
- Etapa 2: Extrair a face da imagem e alinhá-la numa posição padronizada.
- Etapa 3: Converter a imagem da face em atributos (conhecidos como impressão facial). Alguns atributos que caracterizam a face de uma pessoa serão extraídos a partir da imagem da face recortada e alinhada.
- Etapa 4: Localizar a correspondência no BD. A impressão facial será utilizada para fazer busca num BD de rostos conhecidos.

Há 15 anos atrás, já havia boas soluções para as etapas 1, 2 e 4 antes mesmo de deep learning, por exemplo, usando respectivamente o algoritmo de Viola e Jones (filtros de Haar e boosting em cascata), transformações geométricas e busca do vizinho mais próximo. Porém, não havia uma boa solução para a etapa 3: extrair a impressão facial.

Nota: Antes de *redes convolucionais* e *aprendizado profundo*, eram usadas técnicas como *EigenFaces*, *FisherFaces* e *LBPH* para extrair a impressão facial. O site abaixo traz um bom tutorial de técnicas clássicas de reconhecimento de faces em OpenCV:
http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html
Os exercícios-programas de 2017 também servem como exemplos de reconhecimento facial antes de deep learning:
http://www.lps.usp.br/hae/psi5796/ep2-2017/index.html
http://www.lps.usp.br/hae/psi3471/ep2-2017/index.html

Como vimos nesta aula, é possível usar pares (ou triplas) de imagens de pessoas iguais e diferentes para treinar uma rede siamesa. A rede treinada irá extrair a “impressão facial”, isto é, um vetor de atributos. As imagens de uma mesma pessoa irão gerar “impressões faciais” semelhante, enquanto que as imagens de pessoas diferentes irão gerar “impressões faciais” diferentes.

2. Biblioteca DeepFace

Existem vários modelos de redes convolucionais gratuitos prontos para serem usados no reconhecimento facial. Aparentemente, *DeepFace* de Sefik Serengil é uma das bibliotecas mais populares (em 2022).

https://pypi.org/project/deepface/
https://github.com/serengil/deepface
https://viso.ai/computer-vision/deepface/
https://sefiks.com/2020/05/01/a-gentle-introduction-to-face-recognition-in-deep-learning/
https://www.youtube.com/playlist?list=PLsS_1RYmYQQFdWqxQggXHynP1rqaYXv_E

Esses sites e vídeos são bem didáticos e é possível usar os modelos gerados por grandes empresas, como Microsoft, Google, Facebook, etc. Serengil faz uma comparação dos diferentes modelos (VGG-Face da Oxford, OpenFace da Carnegie Mellon, DeepFace da Facebook e FaceNet da Google) no vídeo abaixo:

https://www.youtube.com/watch?v=i_MOwvhbLdI

O reconhecimento facial depende da extração de "impressão facial" a partir das imagens das faces. A comparação entre duas impressões faciais (dois vetores de alta dimensão) pode ser feita calculando:

1. Cosseno do ângulo α formado pelos dois vetores ou
2. Distância euclidiana *d* entre os dois vetores.

Também poderia usar a diferença absoluta, como fizemos em OmniGlot.

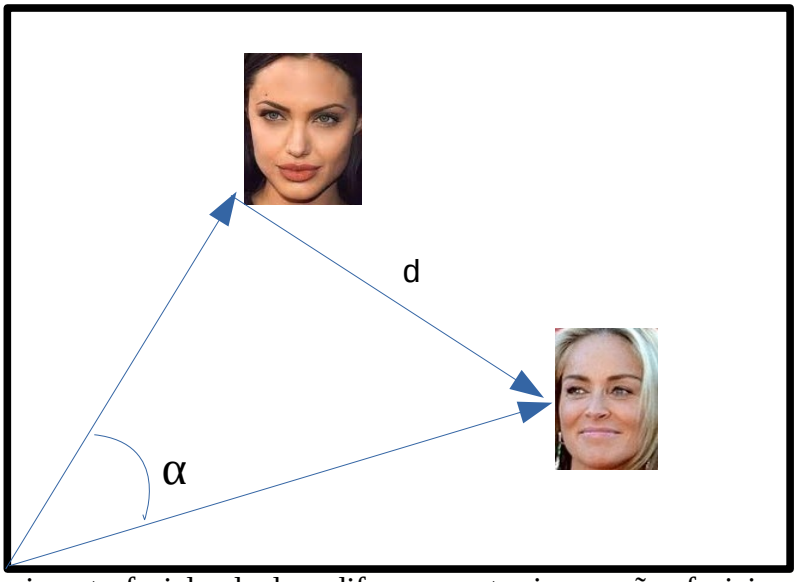


Figura 11: Reconhecimento facial calcula a diferença entre impressões faciais calculando a distância euclidiana *d* ou cosseno dos ângulos α.

Serengil testou vários os modelos e métricas de distância, chegando à conclusão de que o modelo com a maior taxa de acerto é FaceNet da Google usando distância euclidiana, com taxa de acerto de 98,57% (em 2022). FaceNet com distância cosseno atinge a segunda maior taxa de acerto 98,21% mas com a vantagem de que a distância estará limitada ao intervalo [-1; +1].

O programa 3 exemplifica o uso de *DeepFace*.

Antes de mais nada, é necessário instalar o módulo DeepFace:

```
pip install deepface     OU
pip3 install deepface
```

```
url='http://www.lps.usp.br/hae/apostila/face.zip'
import os; nomeArq=os.path.split(url)[1]
if not os.path.exists(nomeArq):
  print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
  os.system("wget -nc -U 'Firefox/50.0' "+url)
else:
  print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
print("Descompactando arquivos novos de",nomeArq)
os.system("unzip -u "+nomeArq)

#!pip3 install deepface
#face1b.py
from deepface import DeepFace
model_name="Facenet"; distance_metric="cosine"

img1_path="sharon_stone1.jpg"; img2_path="sharon_stone2.jpg"
resp = DeepFace.verify (img1_path=img1_path, img2_path=img2_path,
  model_name=model_name,distance_metric=distance_metric)
print("Comparando",img1_path,"com",img2_path,":")
print(resp,"\n")

img1_path="angelina_jolie1.jpg"; img2_path="angelina_jolie2.jpg"
resp = DeepFace.verify (img1_path=img1_path, img2_path=img2_path,
  model_name=model_name,distance_metric=distance_metric)
print("Comparando",img1_path,"com",img2_path,":")
print(resp,"\n")

img1_path="angelina_jolie1.jpg"; img2_path="sharon_stone1.jpg"
resp = DeepFace.verify (img1_path=img1_path, img2_path=img2_path,
  model_name=model_name,distance_metric=distance_metric)
print("Comparando",img1_path,"com",img2_path,":")
print(resp,"\n")

img1_path="angelina_jolie1.jpg"; img2_path="sharon_stone2.jpg"
resp = DeepFace.verify (img1_path=img1_path, img2_path=img2_path,
  model_name=model_name,distance_metric=distance_metric)
print("Comparando",img1_path,"com",img2_path,":")
print(resp,"\n")

img1_path="angelina_jolie2.jpg"; img2_path="sharon_stone1.jpg"
resp = DeepFace.verify (img1_path=img1_path, img2_path=img2_path,
  model_name=model_name,distance_metric=distance_metric)
print("Comparando",img1_path,"com",img2_path,":")
print(resp,"\n")

img1_path="angelina_jolie2.jpg"; img2_path="sharon_stone2.jpg"
resp = DeepFace.verify (img1_path=img1_path, img2_path=img2_path,
  model_name=model_name,distance_metric=distance_metric)
print("Comparando",img1_path,"com",img2_path,":")
print(resp,"\n")
```

Programa 3: Exemplo de uso de DeepFace. ~/deep/algpi/face/face1b.py
https://colab.research.google.com/drive/1yL3KZnKZOj929htCIT66j2HuqVHQ8zat?usp=sharing

Esse programa baixa 4 imagens no diretório local:


sharon_stone1.jpg


sharon_stone2.jpg


angelina_jolie1.jpg


angelina_jolie2.jpg

Figura 12: Faces usadas para testar DeepFace.

Depois, compara todas as pares de imagens, obtendo:

```
Comparando sharon_stone1.jpg com sharon_stone2.jpg :
{'verified': True, 'distance': 0.2302723612803883, 'threshold': 0.4, 'model': 'Facenet',
'detector_backend': 'opencv', 'similarity_metric': 'cosine'}

Comparando angelina_jolie1.jpg com angelina_jolie2.jpg :
{'verified': False, 'distance': 0.8713807300927271, 'threshold': 0.4, 'model': 'Facenet',
'detector_backend': 'opencv', 'similarity_metric': 'cosine'}

Comparando angelina_jolie1.jpg com sharon_stone1.jpg :
{'verified': False, 'distance': 0.9549059975525636, 'threshold': 0.4, 'model': 'Facenet',
'detector_backend': 'opencv', 'similarity_metric': 'cosine'}

Comparando angelina_jolie1.jpg com sharon_stone2.jpg :
{'verified': False, 'distance': 1.0752758541684009, 'threshold': 0.4, 'model': 'Facenet',
'detector_backend': 'opencv', 'similarity_metric': 'cosine'}

Comparando angelina_jolie2.jpg com sharon_stone1.jpg :
{'verified': False, 'distance': 0.9155319732634719, 'threshold': 0.4, 'model': 'Facenet',
'detector_backend': 'opencv', 'similarity_metric': 'cosine'}

Comparando angelina_jolie2.jpg com sharon_stone2.jpg :
{'verified': False, 'distance': 0.8295508499244636, 'threshold': 0.4, 'model': 'Facenet',
'detector_backend': 'opencv', 'similarity_metric': 'cosine'}
```

Os acertos estão marcados em verde e o único erro em vermelho. Repare que o programa errou na comparação entre *angelina_jolie1* e *angelina_jolie2*, dizendo que as fotos são de pessoas diferentes. O modelo está longe de ser perfeito...

Esta biblioteca possui a função *find*, que acha rapidamente as faces mais parecidas com a face de busca num BD de faces.

Exercício: Repita os testes do programa 3 escolhendo outras 4 imagens (um par de imagens de uma pessoa A e outro par de imagens de uma pessoa B).

3. Exercício-programa de PSI5790 de 2023

No site abaixo:
https://www.kaggle.com/datasets/rawatjitesh/avengers-face-recognition
há um conjunto de 274 imagens faciais dos 5 atores de vingadores. Um exercício de 2023:
http://www.lps.usp.br/hae/psi5790/ep1-2023/index.html
pedia para reconhecer os rostos de 5 atores de vingadores num BD com poucas amostras (50 imagens de cada ator).

Figura 13: Exemplos de imagens faciais dos 5 atores de vingadores.

Os resultados obtidos por mim foram:
1) Usando CNN convencional treinada do zero para classificar imagens, obtive erro médio de 30%.
2) Usando DeepFace, obtive erro médio de 1,2%.

Isto mostra a vantagem de usar uma rede pré-treinada para reconhecer faces.

[PSI3472 aula 11/12. Lição de casa #1 de 2. Vale 5,0]
Baixe o conjunto de imagens faciais dos 5 vingadores de:
https://www.kaggle.com/datasets/rawatjitesh/avengers-face-recognition ou
http://www.lps.usp.br/hae/temp/avengers.zip

Escolha 5 imagens (1 imagem de cada vingador) como imagens de suporte. Escolha outras 5 imagens (1 imagem de cada vingador) como imagens de consulta (teste).

Utilizando biblioteca DeepFace, verifique quantas faces (das 5 de consulta) o modelo consegue identificar (classificar) corretamente. Isto é, compare cada imagem de consulta com as 5 imagens de suporte e escolha a imagem de suporte de menor distância. Isto seria um problema 5-way 1-shot learning, conforme definido na seção II, pg. 4 desta apostila. Qual foi a taxa de acerto?

V. Few-shot learning

[https://www.youtube.com/watch?v=U6uFOIURcD0]

Para problemas com *K*-shot, com *K* relativamente grande. há soluções melhores do que as técnicas que descrevemos acima.

[Em edição]

VI. Zero-shot learning

https://cetinsamet.medium.com/zero-shot-learning-53080995d45f
https://medium.com/@tenyks_blogger/zero-shot-ai-the-end-of-fine-tuning-as-we-know-it-1e9f0215967e
https://www.v7labs.com/blog/zero-shot-learning-guide#:~:text=Zero%2DShot%20Learning%20models%20are,classify%20birds%20on%20the%20fly.
https://blog.roboflow.com/zero-shot-learning-computer-vision/
https://cvml.ista.ac.at/AwA2/

[PSI3472 aula 11, Fim]

Referências:

[Lamba2019] One Shot Learning with Siamese Networks Using Keras. https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d

[Brownlee2019] One-Shot Learning for Face Recognition. https://machinelearningmastery.com/one-shot-learning-with-siamese-networks-contrastive-and-triplet-loss-for-face-recognition/

[Bouma2017] One Shot Learning and Siamese Networks in Keras. https://sorenbouma.github.io/blog/oneshot/

[Lake2019] Breden Lake, Omniglot data set for one-shot learning. https://github.com/brendenlake/omniglot

[Jade2022] Jade Vaibhav, Signature-verification-using-deep-learning. https://github.com/jadevaibhav/Signature-verification-using-deep-learning/blob/master/README.md

Vídeos muito bons sobre few-shot learning, incluindo one-shot learning:
Few-Shot Learning Shusen Wang
https://www.youtube.com/watch?v=hE7eGew4eeg
https://www.youtube.com/watch?v=4S-XDefSjTM
https://www.youtube.com/watch?v=U6uFOIURcD0

Conceitos gerais de few-shot learning:
Everything You Need To Know About Few-Shot Learning Rohit Kundu
https://blog.paperspace.com/few-shot-learning/
Fornece vários exemplos de aplicações. Diz que reconhecimento facial usa one-shot learning. Mas como as descrições são curtas, não dá para entender.

Curso de Stanford sobre multi-task and meta learning:
Stanford CS330: Deep Multi-Task and Meta Learning I Autumn 2022
https://www.youtube.com/playlist?list=PLoROMvodv4rNjRoawgt72BBNwL2V7doGI

Exemplo em Keras para Omniglot (Reptile) com notebook Colab que funciona:
https://keras.io/examples/vision/reptile
Notebook colab correspondente:
https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/vision/ipynb/reptile.ipynb
Nota: Não consigo entender o que o resultado representa. Não explica qual é a métrica de custo.

Meta-Learning para Omniglot (MANN).
Meta-Learning for Omniglot Character Few-Shot Learning Classification using MANN Shihab Aaqil Ahamed
https://medium.com/@shihabaaqilahamed/meta-learning-for-omniglot-character-few-shot-learning-classification-using-mann-17271c0cb61c
Usa LSTM e 5 way 1 e chega a 99,5% de “acuracidade de treino” (ou seria de teste)?
Tem código Tensorflow, mas parece incompleto.

Outro exemplo em Keras para Omniglot e mini-imagenet usando Prototypical Network:
Re-implementation of the Prototypical Network for Few-Shot Learning using Tensorflow 2.0 + Keras napat thumvanit
https://medium.com/@barnrang/re-implementation-of-the-prototypical-network-for-few-shot-learning-using-tensorflow-2-0-keras-b2adac8e49e0
Obtém acuracidade:
- 98,5% em 5-ways 1-shot
- 99,7% em 5-ways 5-shot

Meta Learning Framework TF 2.0
Há vários exemplos de meta learning em TF 2.0 com datasets
https://github.com/siavash-khodadadeh/MetaLearning-TF2.0

Explicação em alto nível. Sem implementação.
The Complete Guide to Few-Shot Learning Ayush Parti
https://pareto.ai/blog/few-shot-learning
As descrições são muito vagas e genéricas. Não serve.

Perguntando “Cite good tutorials and blogs about few-shot learning.” para ChatGPT, devolveu vários links. A maioria dos links estava quebrado. Um que funcionou:

Meta-Learning: Learning to Learn Fast Lilia Weng

https://lilianweng.github.io/posts/2018-11-30-meta-learning/

Descreve várias técnicas de meta-learning, mas em forma de overview.

Livro sobre Meta Learning:

Hands-On Meta Learning with Python: Meta learning using one-shot learning, MAML, Reptile, and Meta-SGD with TensorFlow
Sudharsan Ravichandiran
https://github.com/sudharsan13296/Hands-On-Meta-Learning-With-Python?tab=readme-ov-file
https://www.amazon.com/Hands-Meta-Learning-Python-algorithms-ebook/dp/B07KJJHYKF

Não está disponível gratuitamente.