

## I. One-Shot Learning

### 1. Introdução

Vamos seguir as explicações dos posts [[Brownlee2019](#), [Lamba2019](#), [Bouma2017](#) ] para aprender sobre One-Shot Learning.

Já vimos que para fazer classificação com CNN é necessária uma grande quantidade de dados de treinamento. Por exemplo, para ensinar CNN a distinguir duas pessoas A e B, são necessárias muitas fotos das pessoas A e B. One-shot learning permite efetuar classificação a partir de poucas ou mesmo uma única imagem de treinamento (uma única foto da pessoa A e outra da pessoa B). Isto parece mágica, mas veremos que na verdade aprendizagem one-shot utilizou previamente uma grande quantidade de imagens 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.

One-shot learning aprende, a partir de uma quantidade grande de fotos de pessoas, o que faz duas fotos serem de uma mesma pessoa. Isto é, aprende uma “função distância” que devolve valor baixo se as duas fotos forem da mesma pessoa e devolve 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 (aliás, há vários modelos gratuitos pré-treinados). Depois, basta inserir uma única foto (ou poucas fotos) de cada funcionário para poder verificar se a pessoa que está sendo fotografada pela câmera de segurança é a mesma pessoa (ou não) daquela armazenada no BD. 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.

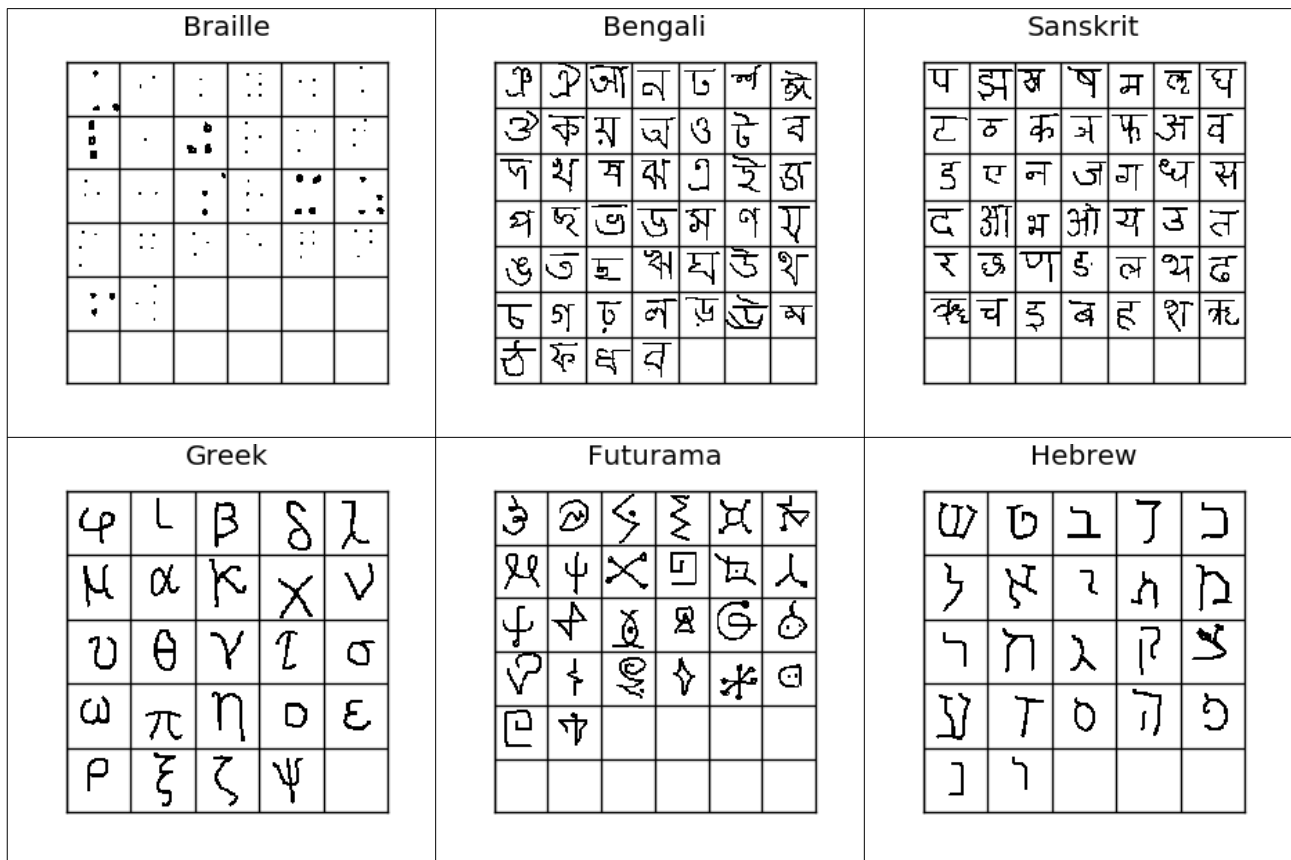
Outro exemplo de uso de “one-shot learning” é na verificação de assinatura. Basta ter poucas assinaturas de cada cliente para que o sistema possa verificar se uma dada assinatura é do cliente registrado (ou não). [[Jade2022](#)]

## 2. Banco de dados Omniglot

Seguindo [[Lamba2019](#), [Bouma2017](#), [Lake2019](#) ], vamos usar o banco de dados (BD) Omniglot para testar one-shot learning. Este BD 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. Há 964 caracteres para o treino e 659 caracteres para o teste e 20 exemplos para caractere.

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 BD de [[Lake2019](#) ], copiando as imagens images\_background.zip (imagens de treino) e images\_evaluation.zip (imagens de teste) do diretório “python”.



Este BD contém 20 imagens para cada caractere. Abaixo, exemplo de 20 imagens de um mesmo caractere:

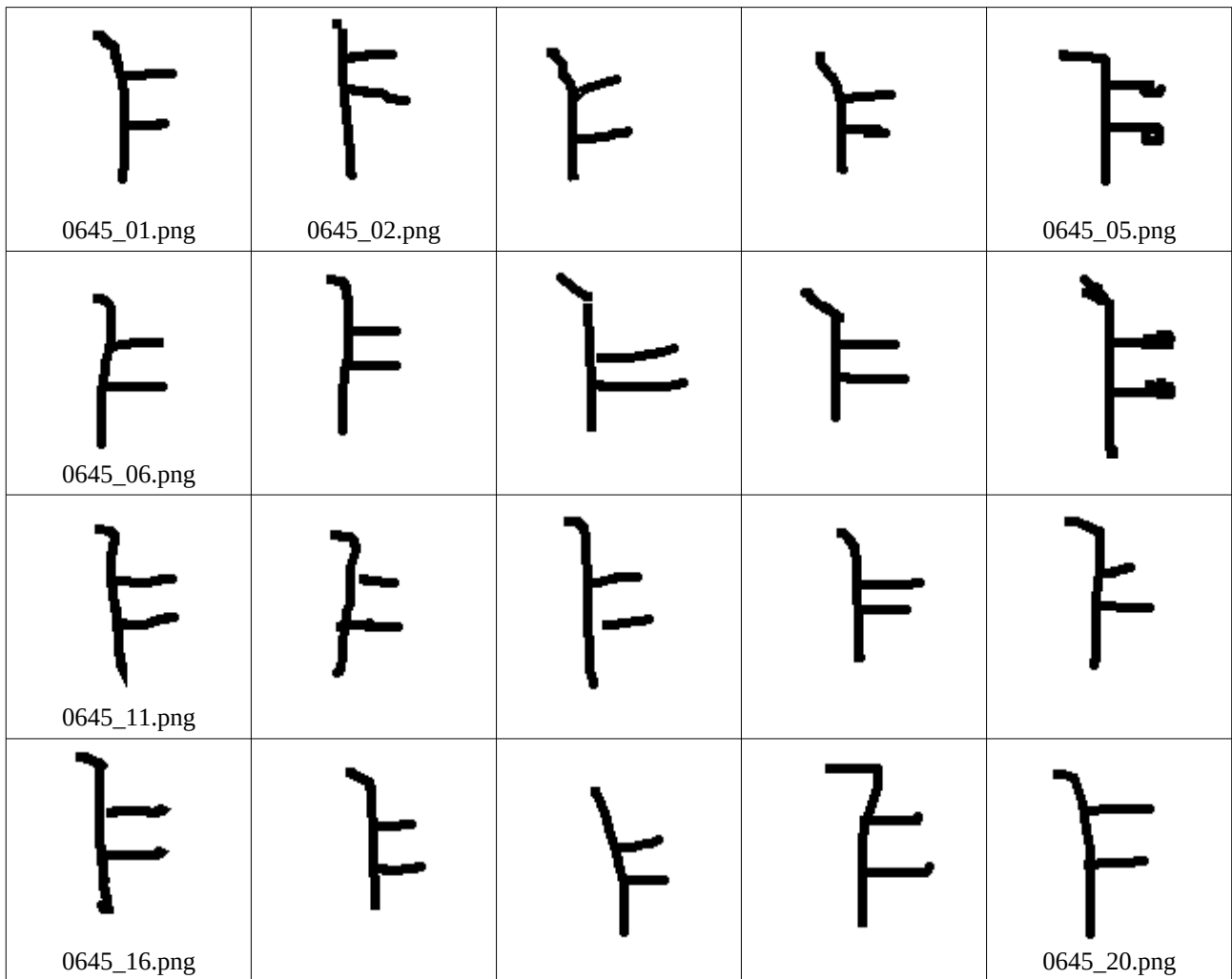


Figura 2: Exemplo de 20 imagens que representam o mesmo caractere.

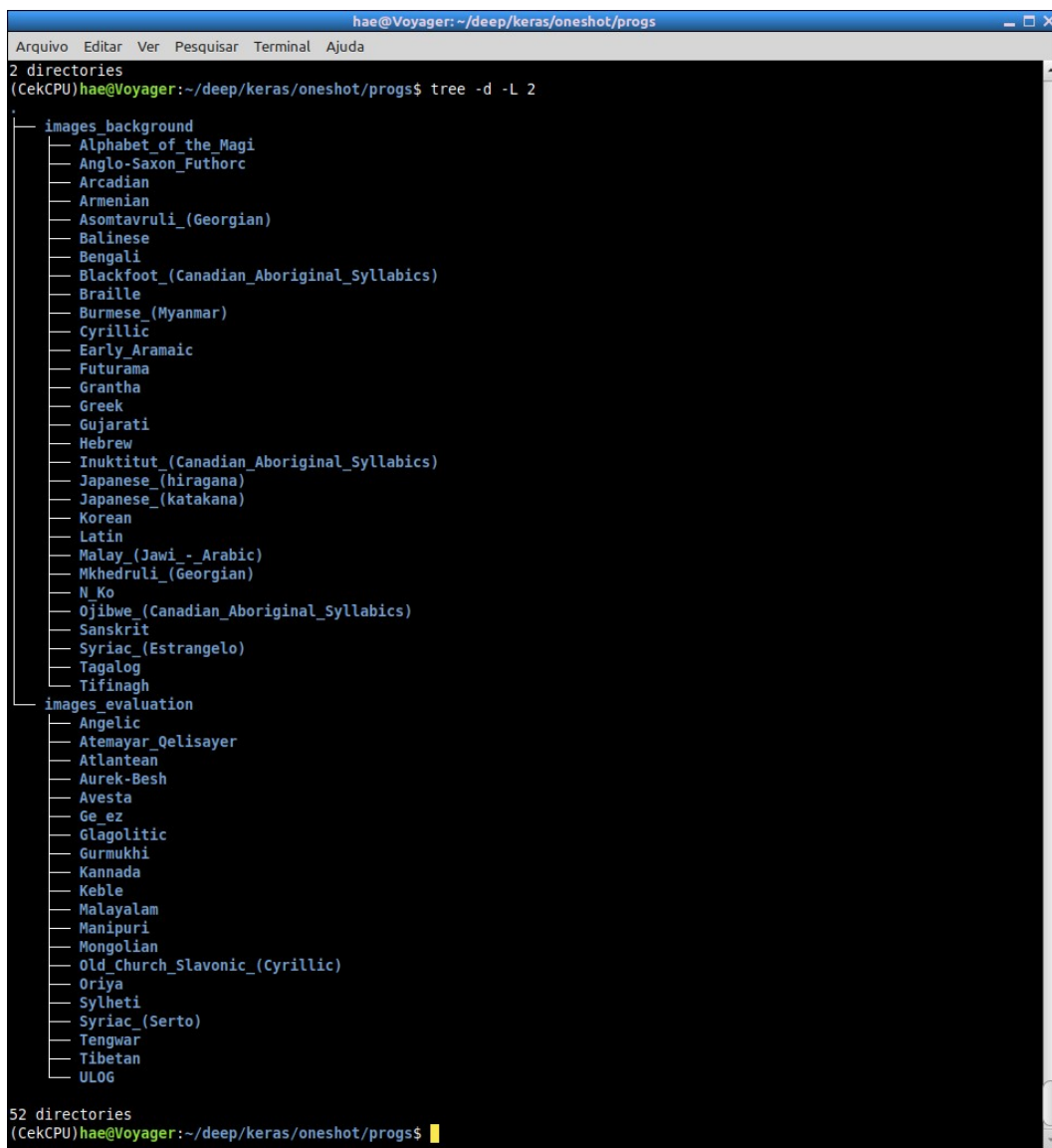
O seguinte trecho de programa baixa e descompacta esse BD no diretório atual (no Colab ou no computador local):

```
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: Faz download do BD

[[https://colab.research.google.com/drive/1Yp4y1\\_SSM8RFFPUnnlpCBhNI1iK0q2R?usp=sharing](https://colab.research.google.com/drive/1Yp4y1_SSM8RFFPUnnlpCBhNI1iK0q2R?usp=sharing)]

Após executar esse trecho, haverá dois subdiretórios no seu diretório atual: `images_background` (imagens de treino) e `images_evaluation` (imagens de teste), cada um com mais subdiretórios.



```
hae@Voyager:~/deep/keras/oneshot/progs
Arquivo Editar Ver Pesquisar Terminal Ajuda
2 directories
(CekCPU)hae@Voyager:~/deep/keras/oneshot/progs$ tree -d -L 2
.
├── images_background
│   ├── Alphabet_of_the_Magi
│   ├── Anglo-Saxon_Futhorc
│   ├── Arcadian
│   ├── Armenian
│   ├── Asomtavruli_(Georgian)
│   ├── Balinese
│   ├── Bengali
│   ├── Blackfoot_(Canadian_Aboriginal_Syllabics)
│   ├── Braille
│   ├── Burmese_(Myanmar)
│   ├── Cyrillic
│   ├── Early_Aramaic
│   ├── Futurama
│   ├── Grantha
│   ├── Greek
│   ├── Gujarati
│   ├── Hebrew
│   ├── Inuktitut_(Canadian_Aboriginal_Syllabics)
│   ├── Japanese_(hiragana)
│   ├── Japanese_(katakana)
│   ├── Korean
│   ├── Latin
│   ├── Malay_(Jawi_-_Arabic)
│   ├── Mkhedruli_(Georgian)
│   ├── N_Ko
│   ├── Ojibwe_(Canadian_Aboriginal_Syllabics)
│   ├── Sanskrit
│   ├── Syriac_(Estrangelo)
│   ├── Tagalog
│   └── Tifinagh
├── images_evaluation
│   ├── Angelic
│   ├── Atemayar_Qelisayer
│   ├── Atlantean
│   ├── Aurek-Besh
│   ├── Avesta
│   ├── Ge_oz
│   ├── Glagolitic
│   ├── Gurmukhi
│   ├── Kannada
│   ├── Keble
│   ├── Malayalam
│   ├── Manipuri
│   ├── Mongolian
│   ├── Old_Church_Slavonic_(Cyrillic)
│   ├── Oriya
│   ├── Sylheti
│   ├── Syriac_(Serto)
│   ├── Tengwar
│   ├── Tibetan
│   └── ULOG
52 directories
(CekCPU)hae@Voyager:~/deep/keras/oneshot/progs$
```

Figura 3: Estrutura do BD OmniGlot.

### 3. One-shot learning

Vamos usar o programa de Harshall Lamba, fazendo adaptações para simplificar o programa e facilitar o entendimento:

<https://github.com/hlamba28/One-Shot-Learning-with-Siamese-Networks>

```
1 """oneshot.ipynb
2 Original file is located at
3 https://colab.research.google.com/drive/1lYp4y1_SSM8RFFPUnnlpCBhNI1iK0q2R
4 """
5
6 """ O código seguinte baixa o BD Omniglot de: https://github.com/brendenlake/omniglot"""
7 import os;
8 lista=['https://github.com/brendenlake/omniglot/raw/master/python/images_background.zip', \
9        'https://github.com/brendenlake/omniglot/raw/master/python/images_evaluation.zip']
10 for url in lista:
11     nomeArq=os.path.split(url)[1]
12     if not os.path.exists(nomeArq):
13         print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
14         os.system("wget -nc -U 'Firefox/50.0' "+url)
15     else:
16         print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
17     print("Desccompactando arquivos novos de",nomeArq)
18     os.system("unzip -u "+nomeArq)
19
20 """O programa abaixo é uma adaptação do programa de Harshall Lamba:
21 https://github.com/hlamba28/One-Shot-Learning-with-Siamese-Networks
22 O artigo original de Harshall Lamba está em:
23 https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d
24 """
25 import sys
26 import numpy as np
27 import pandas as pd
28 from matplotlib.pyplot import imread
29 import pickle, os, cv2, time
30 import matplotlib.pyplot as plt
31
32 import tensorflow as tf
33 from tensorflow.keras.models import Sequential, load_model
34 from tensorflow.keras.optimizers import Adam
35 from tensorflow.keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate
36 import tensorflow.keras.initializers as initializers
37 from tensorflow.keras.models import Model
38 from tensorflow.keras.layers import BatchNormalization, MaxPooling2D, Concatenate
39 from tensorflow.keras.layers import Layer, Lambda, Flatten, Dense
40 from tensorflow.keras.initializers import glorot_uniform
41 from tensorflow.keras.regularizers import l2
42 from tensorflow.keras import backend as K
43 from sklearn.utils import shuffle
44 import numpy.random as rng
45
46 """Coloque abaixo os diretórios do seu ambiente de trabalho"""
47 train_folder = "./images_background/"
48 val_folder = './images_evaluation/'
49 save_path = './'
50 model_path = './'
51
52 # Funcao para carregar imagens
53 def loadimgs(path,n = 0):
54     #path => Path of train directory or test directory
55     X=[]; y=[]
56     cat_dict = {}; lang_dict = {}; curr_y = n
57     # we load every alphabet separately so we can isolate them later
58     for alphabet in os.listdir(path):
59         print("loading alphabet: " + alphabet)
60         lang_dict[alphabet] = [curr_y, None]
61         alphabet_path = os.path.join(path,alphabet)
62         for letter in os.listdir(alphabet_path):
63             cat_dict[curr_y] = (alphabet, letter)
64             category_images=[]
65             letter_path = os.path.join(alphabet_path, letter)
66             # read all the images in the current category
67             for filename in os.listdir(letter_path):
68                 image_path = os.path.join(letter_path, filename)
69                 image = imread(image_path)
70                 category_images.append(image)
71             y.append(curr_y)
72         try:
73             X.append(np.stack(category_images))
74         # edge case - last one
75         except ValueError as e:
76             print(e); print("error - category_images:", category_images)
77             curr_y += 1
78             lang_dict[alphabet][1] = curr_y - 1
79     y = np.vstack(y); X = np.stack(X)
80     return X,y, lang_dict
81
82 # Carrega as imagens de treino
83 Xtrain,ytrain,train_classes=loadimgs(train_folder)
84
85 # Carrega as imagens de teste
86 Xval,yval,val_classes=loadimgs(val_folder)
87
88 # Verifica o formato dos dados lidos
89 print(Xtrain.shape)
90 print(Xtrain[0,0,70])
91 print(ytrain.shape)
92 print(ytrain[0:5])
```

```

93 print(train_classes)
94 print(Xval.shape)
95
96 # Funcao que constroi a rede siamesa
97 def get_siamese_model(input_shape):
98     #Model architecture based on the one provided in: http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf
99     left_input = Input(input_shape); right_input = Input(input_shape)
100
101     seq = Sequential()
102     seq.add(Conv2D(64, (10,10), activation='relu', input_shape=input_shape,
103                 kernel_initializer=initializers.RandomNormal(stddev=1e-2), kernel_regularizer=l2(2e-4)))
104     seq.add(MaxPooling2D())
105     seq.add(Conv2D(128, (7,7), activation='relu',
106                 kernel_initializer=initializers.RandomNormal(stddev=1e-2),
107                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2), kernel_regularizer=l2(2e-4)))
108     seq.add(MaxPooling2D())
109     seq.add(Conv2D(128, (4,4), activation='relu', kernel_initializer=initializers.RandomNormal(stddev=1e-2),
110                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2), kernel_regularizer=l2(2e-4)))
111     seq.add(MaxPooling2D())
112     seq.add(Conv2D(256, (4,4), activation='relu', kernel_initializer=initializers.RandomNormal(stddev=1e-2),
113                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2), kernel_regularizer=l2(2e-4)))
114     seq.add(Flatten())
115     seq.add(Dense(4096, activation='sigmoid',
116                 kernel_regularizer=l2(1e-3),
117                 kernel_initializer=initializers.RandomNormal(stddev=1e-2),
118                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2)))
119
120     # Generate the encodings (feature vectors) for the two images
121     encoded_l = seq(left_input); encoded_r = seq(right_input)
122
123     # Add a customized layer to compute the absolute difference between the encodings
124     L1_distance = K.abs(encoded_l-encoded_r)
125
126     # Add a dense layer with a sigmoid unit to generate the distance score
127     prediction = Dense(1, activation='sigmoid',
128                       bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2))(L1_distance)
129     siamese_net = Model(inputs=[left_input, right_input], outputs=prediction)
130
131     # return the model
132     return siamese_net, seq
133
134 # Execute esta celula para construir a rede "do zero" e imprimir a arquitetura
135 model, seq = get_siamese_model((105, 105, 1))
136 model.summary()
137 seq.summary()
138 from tensorflow.keras.utils import plot_model
139 plot_model(model, to_file=os.path.join(save_path, "siamese_net.png"))
140 plot_model(seq, to_file=os.path.join(save_path, "seq_net.png"))
141 optimizer = Adam(learning_rate = 0.00006)
142 model.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=['accuracy'])
143
144 """
145 # Execute esta celula para carregar a rede ja treinada
146 import os; import gdown
147 url="https://drive.google.com/uc?id=1PAyiw-0_dyti5Fdp_qwgQRqGen4GwY7q"
148 nomeArq="oneshot.h5"
149 if not os.path.exists(nomeArq):
150     print("Baixando o arquivo", nomeArq, "para diretorio default", os.getcwd())
151     gdown.download(url, nomeArq, quiet=False)
152 else:
153     print("O arquivo", nomeArq, "ja existe no diretorio default", os.getcwd())
154     model=load_model("oneshot.h5")
155 """
156
157 # Funcao que pega uma lote de pares de imagens
158 # Metade dos pares de classes diferentes e outra metade dos pares de classes iguais
159 def get_batch(batch_size, s="train"):
160     """Create batch of n pairs, half same class, half different class"""
161     if s == 'train':
162         X = Xtrain; categories = train_classes; replace=False
163     else:
164         X = Xval; categories = val_classes; replace=True
165     n_classes, n_examples, w, h = X.shape
166
167     categories = rng.choice(n_classes, size=(batch_size,), replace=replace)
168     pairs=np.zeros((batch_size, h, w, 1)) for i in range(2))
169     targets=np.ones((batch_size,))
170     targets[batch_size//2:] = 0
171     for i in range(batch_size):
172         category = categories[i]
173         idx_1 = rng.randint(0, n_examples)
174         pairs[0][i, :, :, :] = X[category, idx_1].reshape(w, h, 1)
175         idx_2 = rng.randint(0, n_examples)
176         # pick images of same class for 1st half, different for 2nd
177         if i >= batch_size // 2:
178             category_2 = category
179         else:
180             # add a random number to the category modulo n classes to ensure 2nd image has a different category
181             category_2 = (category + rng.randint(1, n_classes)) % n_classes
182         pairs[1][i, :, :, :] = X[category_2, idx_2].reshape(w, h, 1)
183     return pairs, targets
184
185 # Escolhe aleatoriamente lote de 4 pares de imagens e imprime
186 # 2 pares da mesma classe e 2 de classes diferentes
187 pairs, targets=get_batch(4)
188 print(pairs[0].shape)
189 print(pairs[1].shape)
190 print(targets.shape)
191 f = plt.figure(figsize=(2,4))
192 for i in range(4):
193     f.add_subplot(4,2,2*i+1)
194     plt.imshow(np.squeeze(pairs[0][i]), cmap="gray")
195     plt.axis("off");
196     f.add_subplot(4,2,2*i+2)
197     plt.imshow(np.squeeze(pairs[1][i]), cmap="gray")
198     plt.axis("off");

```

```

199 plt.text(0, 2,int(targets[i]),color="r")
200 plt.savefig("onshot_pares.png")
201 plt.show()
202
203 # Faz o treino ou continua o treino de onde parou.
204 batch_size = 32
205 n_iter = 20000 # No. of training iterations
206
207 print("Starting training process!")
208 print("-----")
209 t_start = time.time()
210 for i in range(n_iter):
211     (inputs,targets) = get_batch(batch_size)
212     loss = model.train_on_batch(inputs, targets)
213     if i%1000==0:
214         print("Epoca=%5d "%(i), "Loss=%f Accuracy=%f"%(loss[0], loss[1]))
215 model.save(os.path.join(model_path, 'onshot.h5'))
216
217 # Gera um lote de teste e verifica a taxa de acerto
218 n=10000
219 qx,qy=get_batch(n, "test")
220 #qp = model.predict(qx)
221 results=model.evaluate(qx,qy)
222 print(results)

```

Programa 1: One shot learning [ [https://colab.research.google.com/drive/1Yp4y1\\_SSM8RFFPUnlpCBhNI1iK0q2R?usp=sharing](https://colab.research.google.com/drive/1Yp4y1_SSM8RFFPUnlpCBhNI1iK0q2R?usp=sharing) ]

### 3.1 Carregar imagens

A função *loadimgs* carrega o BD. Após ter carregado as imagens, executando:

```

Xtrain,ytrain,train_classes=loadimgs(train_folder)
print(Xtrain.shape)
print(Xtrain[0,0,70])
print(ytrain.shape)
print(ytrain[0:41])
print(train_classes)

```

Obtemos a seguinte saída:

```

Xtrain.shape (964, 20, 105, 105)
Xtrain [1. 1. 1. 1. 1. (...) 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. (...)]
ytrain.shape (19280, 1)
ytrain [[0] [0] [0] (...) [0] [1] [1] (...) [1] [2] (...)]
train_classes {'Syriac_(Estrangelo)': [0, 22], 'Mkhedruli_(Georgian)': [23, 63], (...)},

```

Indicando que tensor *Xtrain* possui 964 entradas (caracteres) e cada entrada com 20 imagens 105×105. Os valores dos pixels das imagens são 0 (preto) ou 1 (branco).

O vetor *ytrain* possui 19280=964×20 entradas que indicam o caractere correspondente a cada imagem. 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.

O dicionário *train\_classes* indica quais caracteres pertencem a cada alfabeto. Os caracteres de 0 a 22 pertencem ao alfabeto 'Syriac\_(Estrangelo)', os caracteres 23 a 63 pertencem ao alfabeto 'Mkhedruli\_(Georgian)', etc. Não usaremos este dado na nossa solução.

### 3.2 Rede siamesa

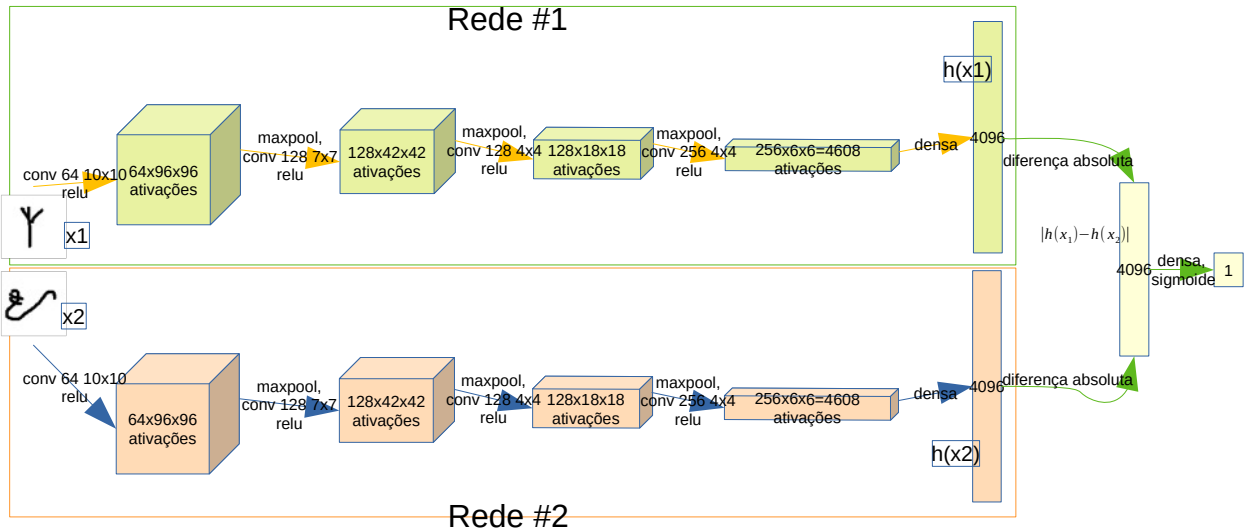


Figura 4: Uma rede siamesa consiste de duas redes completamente iguais, inclusive os parâmetros (pesos e vieses iguais).

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, ilustrada na figura acima. Na verdade, não temos duas redes mas uma única rede que recebe duas imagens diferentes e gera duas saídas diferentes (correspondentes às duas imagens de entrada). Assim, talvez a melhor representação seja a figura abaixo.

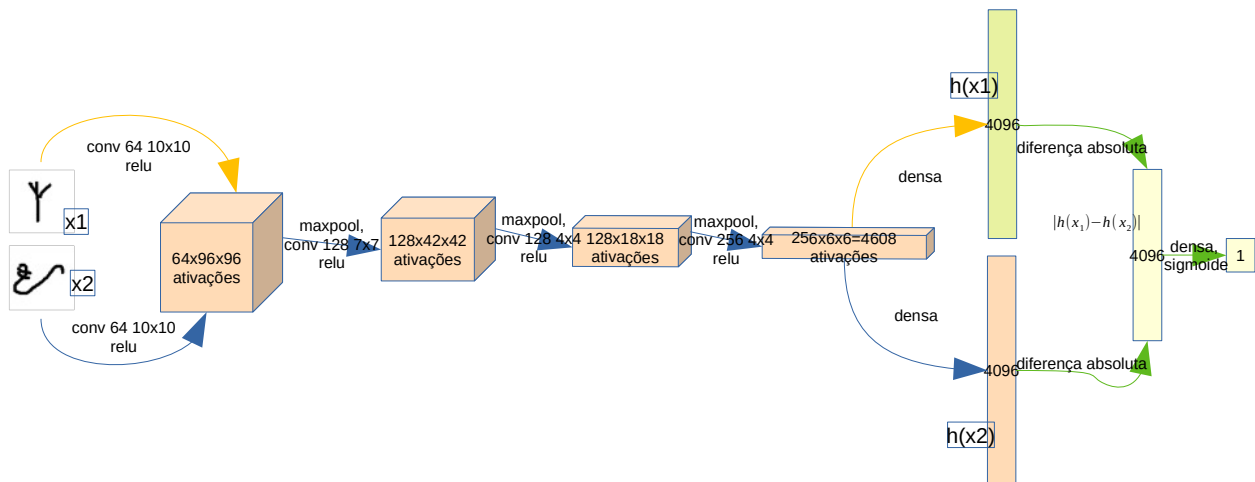


Figura 5: Na verdade, uma rede siamesa consiste de uma única rede utilizada para processar duas imagens diferentes.

Na figura acima, as duas imagens de entrada  $x_1$  e  $x_2$  passam pela mesma rede, gerando dois vetores de atributos  $h(x_1)$  e  $h(x_2)$ . A rede calcula a diferença absoluta  $|h(x_1) - h(x_2)|$  entre os dois vetores, gerando o vetor de diferenças absolutas. O vetor de diferenças 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.



A função `get_siamese_model` constrói e devolve a rede `siamese_net`. Além disso, devolve também a rede `seq`, a parte da rede que é compartilhada para processar as duas imagens. Copio abaixo essa função.

```

97 def get_siamese_model(input_shape):
98     left_input = Input(input_shape)
99     right_input = Input(input_shape)
100
101     # Convolutional Neural Network
102     seq = Sequential()
103     seq.add(Conv2D(64, (10,10), activation='relu', input_shape=input_shape,
104                 kernel_initializer=initializers.RandomNormal(stddev=1e-2), kernel_regularizer=l2(2e-4)))
105     seq.add(MaxPooling2D())
106     seq.add(Conv2D(128, (7,7), activation='relu',
107                 kernel_initializer=initializers.RandomNormal(stddev=1e-2),
108                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2), kernel_regularizer=l2(2e-4)))
109     seq.add(MaxPooling2D())
110     seq.add(Conv2D(128, (4,4), activation='relu', kernel_initializer=initializers.RandomNormal(stddev=1e-2),
111                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2), kernel_regularizer=l2(2e-4)))
112     seq.add(MaxPooling2D())
113     seq.add(Conv2D(256, (4,4), activation='relu', kernel_initializer=initializers.RandomNormal(stddev=1e-2),
114                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2), kernel_regularizer=l2(2e-4)))
115     seq.add(Flatten())
116     seq.add(Dense(4096, activation='sigmoid',
117                 kernel_regularizer=l2(1e-3),
118                 kernel_initializer=initializers.RandomNormal(stddev=1e-2),
119                 bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2)))
120
121     # Generate the encodings (feature vectors) for the two images
122     encoded_l = seq(left_input)
123     encoded_r = seq(right_input)
124
125     L1_distance = K.abs(encoded_l-encoded_r)
126
127     prediction = Dense(1,activation='sigmoid',
128                     bias_initializer=initializers.RandomNormal(mean=0.5, stddev=1e-2))(L1_distance)
129
130     siamese_net = Model(inputs=[left_input,right_input],outputs=prediction)
131
132     return siamese_net, seq

```

Programa 2: Trecho do programa 1 que constrói a rede siamesa.

Essa função constrói a rede `seq` compartilhada (linhas 102-119). Essa rede processa as duas imagens e calcula a diferença absoluta (L1).

Depois de treinada, é possível usar esta rede de duas formas:

- Dadas duas imagens de dois caracteres, calcular a medida da diferença entre elas usando a rede `siamese_net`.
- Dada uma imagem de um caractere, calcular os seus atributos usando a sub-rede `seq`.

Lembrete para mim: Para próximo ano, diminuir a quantidade de parâmetros das redes densas. Experimentar outras estruturas de redes.  
Lembrete: Retirar a última camada densa para comparar diretamente atributos .

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 96, 96, 64)	6464
max_pooling2d (MaxPooling2D)	(None, 48, 48, 64)	0
conv2d_1 (Conv2D)	(None, 42, 42, 128)	401536
max_pooling2d_1 (MaxPooling2D)	(None, 21, 21, 128)	0
conv2d_2 (Conv2D)	(None, 18, 18, 128)	262272
max_pooling2d_2 (MaxPooling2D)	(None, 9, 9, 128)	0
conv2d_3 (Conv2D)	(None, 6, 6, 256)	524544
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 4096)	37.752.832
=====		
Total params: 38,947,648		
Trainable params: 38,947,648		
Non-trainable params: 0		

Figura 6: Rede seq

Model: "model"			
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 105, 105, 1	0	[]
input_2 (InputLayer)	[(None, 105, 105, 1	0	[]
sequential (Sequential)	(None, 4096)	38947648	['input_1[0][0]', 'input_2[0][0]']
tf.math.subtract (TFOpLambda)	(None, 4096)	0	['sequential[0][0]', 'sequential[1][0]']
tf.math.abs (TFOpLambda)	(None, 4096)	0	['tf.math.subtract[0][0]']
dense_1 (Dense)	(None, 1)	4097	['tf.math.abs[0][0]']
=====			
Total params: 38,951,745			
Trainable params: 38,951,745			
Non-trainable params: 0			

Figura 7: Rede siamese\_net

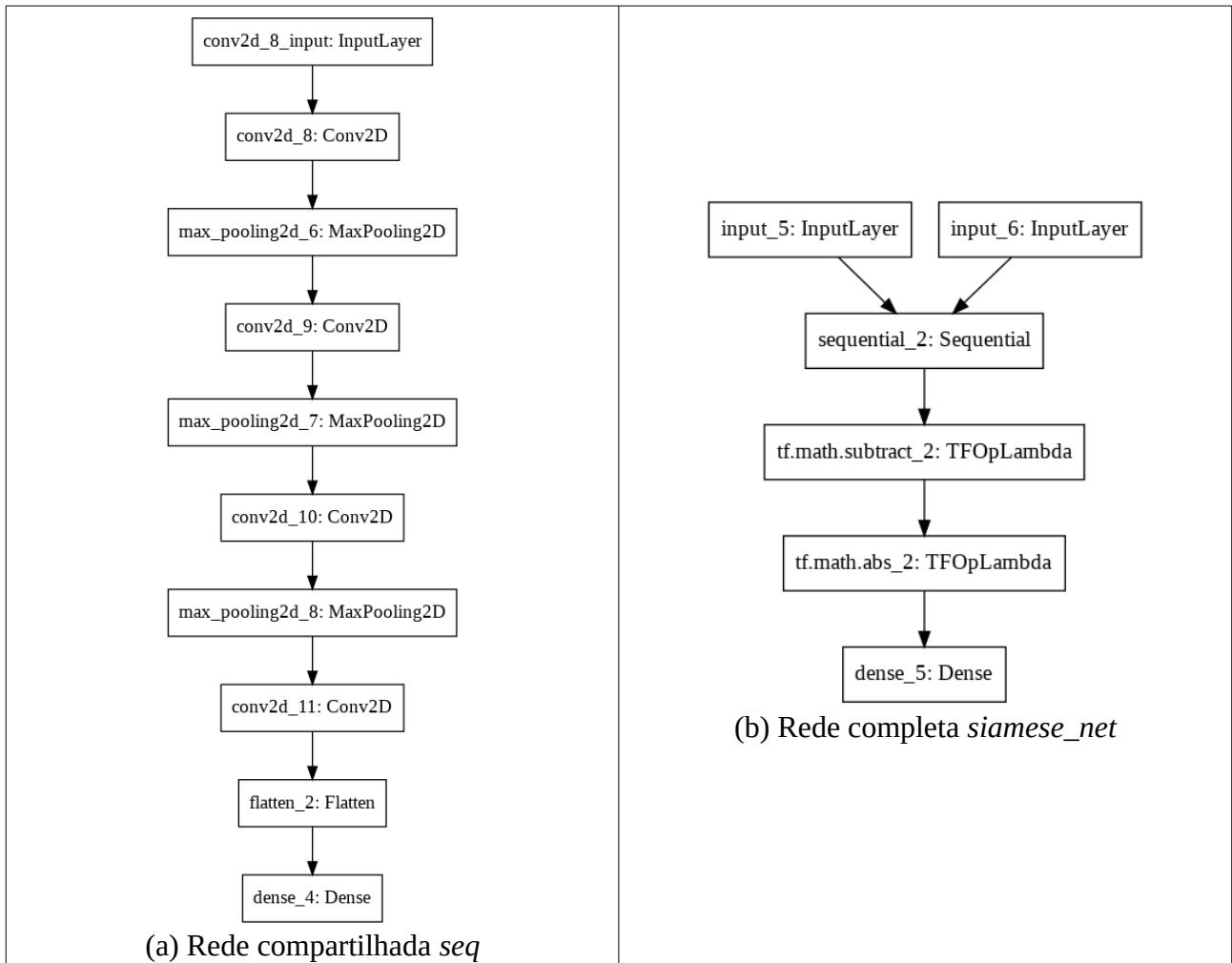


Figura 8: (a) Rede sequencial seq comum às duas entradas. (b) Rede siamesa completa siamese\_net, que utiliza a rede sequencial seq para processar as duas imgs.

### 3.3 Gerar pares de imagens

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”:



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

Vamos alimentar a rede siamesa com lotes de pares de imagens como acima.

### 3.4 Treinar o modelo

Treinando o modelo, obtemos saída como:

```
Epoca= 0 Loss=4.495485 Accuracy=0.500000
Epoca= 1000 Loss=0.818139 Accuracy=0.687500
Epoca= 2000 Loss=0.417647 Accuracy=0.812500
Epoca= 3000 Loss=0.357279 Accuracy=0.906250
Epoca= 4000 Loss=0.377392 Accuracy=0.875000
Epoca= 5000 Loss=0.350694 Accuracy=0.875000
Epoca= 6000 Loss=0.235795 Accuracy=0.937500
Epoca= 7000 Loss=0.368585 Accuracy=0.843750
Epoca= 8000 Loss=0.262728 Accuracy=0.937500
Epoca= 9000 Loss=0.280405 Accuracy=0.968750
Epoca=10000 Loss=0.237525 Accuracy=0.968750
Epoca=11000 Loss=0.181315 Accuracy=0.968750
Epoca=12000 Loss=0.339775 Accuracy=0.875000
Epoca=13000 Loss=0.296000 Accuracy=0.906250
Epoca=14000 Loss=0.193697 Accuracy=0.968750
Epoca=15000 Loss=0.236511 Accuracy=0.968750
Epoca=16000 Loss=0.322566 Accuracy=0.968750
Epoca=17000 Loss=0.266656 Accuracy=0.968750
Epoca=18000 Loss=0.149310 Accuracy=1.000000
Epoca=19000 Loss=0.219586 Accuracy=0.968750
```

Indicando que a função de perda diminui e acuracidade aumenta ao longo das épocas.

### 3.5 Testar o modelo

Depois de treinar a rede, temos um modelo 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, Lamba [Lamba2019] faz um teste complexo chamado “N-way one-shot learning”. Vamos fazer um teste muito mais simples que também consegue mostrar a efetividade do *one-shot learning*. Vamos

gerar aleatoriamente  $n$  pares de imagens, metade representando caracteres iguais e outra metade caracteres diferentes. Depois, vamos alimentar esses pares de imagens à rede siamesa. Consideraremos que, se a resposta da rede for menor que 0,5, a rede classificou o par como caracteres iguais. Fazendo isso com 10000 pares de imagens, obtive taxa de acerto 89,41%.

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

*Exercício:* Escreva um programa que testa “16-way one shot learning”. O problema é calcular, dados um caractere QX à esquerda e mais 16 caracteres à direita (figura abaixo), qual dos 16 caracteres à direita corresponde à mesma classe que QX. Só há uma única resposta correta.

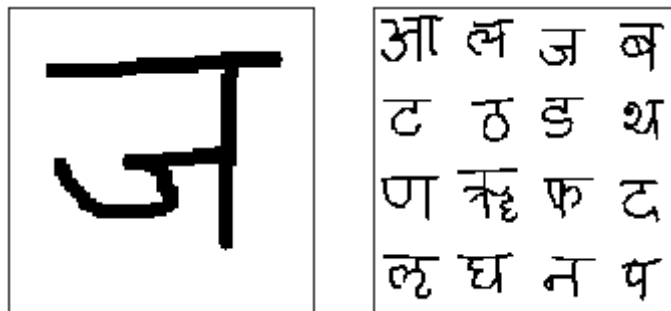


Figura 10: 16-way one shot learning.

Para isso:

a) Carregue o modelo oneshot.h5 já treinado com o programa 1 (oneshot.py):

[https://drive.google.com/file/d/1PAyiW-0\\_dyti5Fdp\\_qwgQRqGen4GwY7q](https://drive.google.com/file/d/1PAyiW-0_dyti5Fdp_qwgQRqGen4GwY7q)

```
# Execute esta celula para carregar a rede ja treinada
import os; import gdown
url="https://drive.google.com/uc?id=1PAyiW-0_dyti5Fdp_qwgQRqGen4GwY7q"
nomeArq="oneshot.h5"
if not os.path.exists(nomeArq):
    print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
    gdown.download(url, nomeArq, quiet=False)
else:
    print("O arquivo",nomeArq,"ja existe no diretorio default",os.getcwd())
model=load_model("oneshot.h5")
```

b) Escolha aleatoriamente um caractere QX e outros 16 caracteres de diferentes classes dos quais apenas um é da mesma classe que QX, como na figura acima.

c) Verifique se o modelo consegue acertar qual é o caractere da mesma classe que QX.

d) Repita os passos (b) e (c) 1000 vezes e verifique a taxa de acerto.

[Em elaboração

*Exercício:* Como vimos, há 964 caracteres para o treino e 659 caracteres para o teste no BD Omniglot, com 20 exemplos para cada caractere. Além disso, disponibilizei o modelo *oneshot.h5* já treinado usando os caracteres de treino pelo programa 1 (oneshot.py):

[https://drive.google.com/file/d/1PAyiW-0\\_dyti5Fdp\\_qwgQRqGen4GwY7q](https://drive.google.com/file/d/1PAyiW-0_dyti5Fdp_qwgQRqGen4GwY7q)

Considerando as imagens 0 a 9 dos 659 caracteres de teste como modelos, classifique as imagens 10 a 19 dos 659 caracteres de teste.

Acho que precisa retirar a última camada densa para que possa fazer este exercício.

]

*Exercício:* Escreva um programa que:

a) Carrega o modelo oneshot.h5 já treinado com o programa 1 (oneshot.py):

[https://drive.google.com/file/d/1PAyiW-0\\_dyti5Fdp\\_qwgQRqGen4GwY7q](https://drive.google.com/file/d/1PAyiW-0_dyti5Fdp_qwgQRqGen4GwY7q)

```
# Execute esta célula para carregar a rede já treinada
import os; import gdown
url="https://drive.google.com/uc?id=1PAyiW-0_dyti5Fdp_qwgQRqGen4GwY7q"
nomeArq="oneshot.h5"
if not os.path.exists(nomeArq):
    print("Baixando o arquivo",nomeArq,"para diretorio default",os.getcwd())
    gdown.download(url, nomeArq, quiet=False)
else:
    print("O arquivo",nomeArq,"já existe no diretorio default",os.getcwd())
model=load_model("oneshot.h5")
```

b) Lê nomes de duas imagens de OmniGlott e mostra as duas imagens na tela, juntamente com a distância (a resposta da rede) entre elas.

c) Teste o seu programa fornecendo 3 pares de imagens de caracteres iguais e 3 pares de imagens de caracteres diferentes. Qual foi a taxa de acerto?

## II. Reconhecimento facial

### 1. Introdução

Hoje em dia, estamos bem acostumados com a identificação das pessoas usando reconhecimento facial. Por exemplo, os telefones celulares podem ser desbloqueados através de reconhecimento facial. O 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> 1.

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.

Aqui, estamos mais interessados na etapa 3 (extração de impressão facial da imagem), pois já conhecemos (mais ou menos) como se pode resolver as etapas 1, 2 e 4 (por exemplo, usando o algoritmo de Viola e Jones, transformações geométricas e busca do vizinho mais próximo).

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](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:

<http://www.lps.usp.br/hae/psi5796/ep2-2017/index.html>

<http://www.lps.usp.br/hae/psi3471/ep2-2017/index.html>

Como vimos no início desta aula, é possível usar pares 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 modelos de redes convolucionais prontos para serem usados. Aparentemente, *DeepFace* de Sefik Serengil é uma das bibliotecas mais populares.

<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\\_1RYmYQQFdWqxOggXHynP1rqaYXv\\_E](https://www.youtube.com/playlist?list=PLsS_1RYmYQQFdWqxOggXHynP1rqaYXv_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](https://www.youtube.com/watch?v=i_MOwvhbLdI)

Conforme já vimos no one-shot learning, reconhecimento facial depende da extração de “impressão facial” a partir das imagens das faces. A comparação entre duas impressões faciais (dois vetores) pode ser feita calculando o ângulo  $\alpha$  ou distância euclidiana  $d$  entre os dois vetores. Também poderia usar a diferença absoluta, como fizemos em OmniGlot.

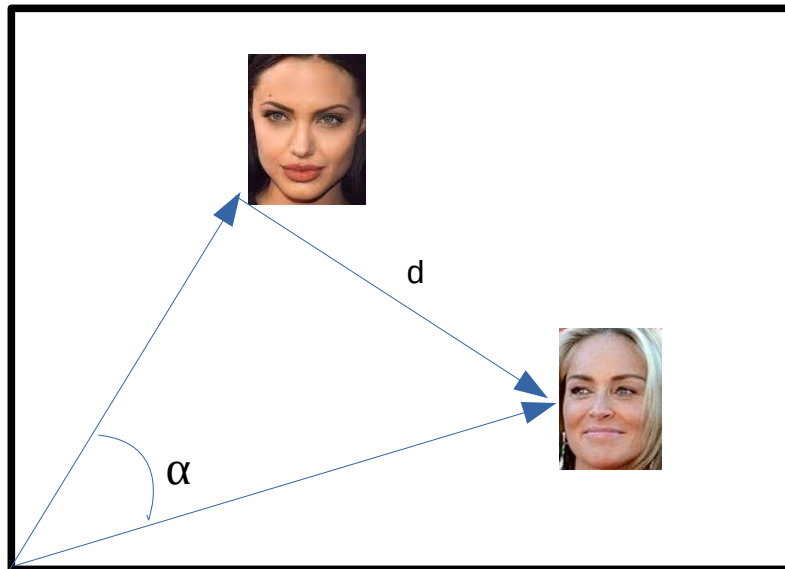


Figura 11: Reconhecimento facial calcula a diferença entre impressões faciais como distância euclidiana  $d$  ou cosseno dos ângulos  $\alpha$ .

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%. 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
```

```

"""face1b.ipynb
  https://colab.research.google.com/drive/1yL3KZnKZOj929htCIT66j2HuqVHQ8zat
"""
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)

#face1b.py
#!pip3 install deepface
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")

```

### Programa 3: Exemplo de uso de DeepFace.

<https://colab.research.google.com/drive/1yL3KZnKZOj929htCIT66j2HuqVHQ8zat?usp=sharing>

Esse programa baixa 4 imagens no diretório local:



Figura 12: Faces usadas para testar DeepFace.



Depois, compara algumas 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'}
```

Os acertos estão marcados em verde e os erros em vermelho. Repare que o programa errou na comparação entre *angelina\_jolie1* e *angelina\_jolie2*, dizendo que as fotos são de pessoas diferentes. Se vê que 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.

### 3. Exemplo do exercício-programa de PSI5790 de 2023

O exercício-programa 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) através de dois métodos:

- 1) Usando qualquer método de classificação de imagens com CNN.
- 2) Usando DeepFace.

Pedia que se fizesse 2-fold cross validation, isto é, usar a metade das imagens para treino e a outra metade para teste, e depois inverter os conjuntos de treino e de teste.



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.

*Exercício:* Pegue 3 fotos suas e 3 fotos de diferentes colegas seus. Para cada uma das 3 fotos suas, ache o rosto mais parecido entre as 5 imagens restantes – se o sistema for bom, deve dizer que o rosto mais parecido é uma das 2 fotos suas e a distância entre as 2 imagens deve ser menor que o threshold, indicando que são fotos da mesma pessoa. Para cada foto do seu colega, procure o rosto

mais parecido entre as 5 imagens restantes – se o sistema for bom, as distâncias da foto para cada um dos 5 rostos deve ser maior do que o threshold, indicando que nenhum dos 5 rostos corresponde à foto.

[PSI3472-2023. Aula 13 parte 1. 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>