Trabalho 3 - Autoencoder Convencional

June 27, 2021

[1]:
```
import tensorflow as tf
import numpy as np
import os
from tqdm.notebook import tqdm
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from PIL import Image
from tensorflow.python.keras.preprocessing import image as kp_image
seed = 42
np.random.seed(seed)
```

1 Trabalho 3 - Autoencoder convencional

Nome: Bruno Rodrigues Silva

Link para as imagens disponíveis neste trabalho: https://drive.google.com/drive/folders/1NMKm-bMNGTPhx2diBXMeIfzgFdtS5CkK?usp=sharing

Descrição: Nesse trabalho você deve criar um autoencoder convencional com camadas convolucionais usando as imagens do conjunto "Anime Faces", que pode ser obtido em https://github.com/bchao1/Anime-Face-Dataset.

Para facilitar o treinamento do autoencoder redimensione as imagens para 64x64x3 pixels.

Após treinar o seu autoencoder realize os seguintes testes:

1) Reconstrução de imagens do conjunto de dados. Reconstrua pelo menos 16 imagens e faça um gráfico das imagens reconstruídas junto com as imagens originais. Além disso, calcule o erro de reconstrução de cada imagem.

2) Criação de imagens novas originais por meio da combinação da representação latente de duas imagens. Nessa etapa crie várias transições entre duas imagens do conjunto de teste, iniciando de uma delas até obter a outra. Faça isso para pelo menos 8 pares de imagens do conjunto de dados.

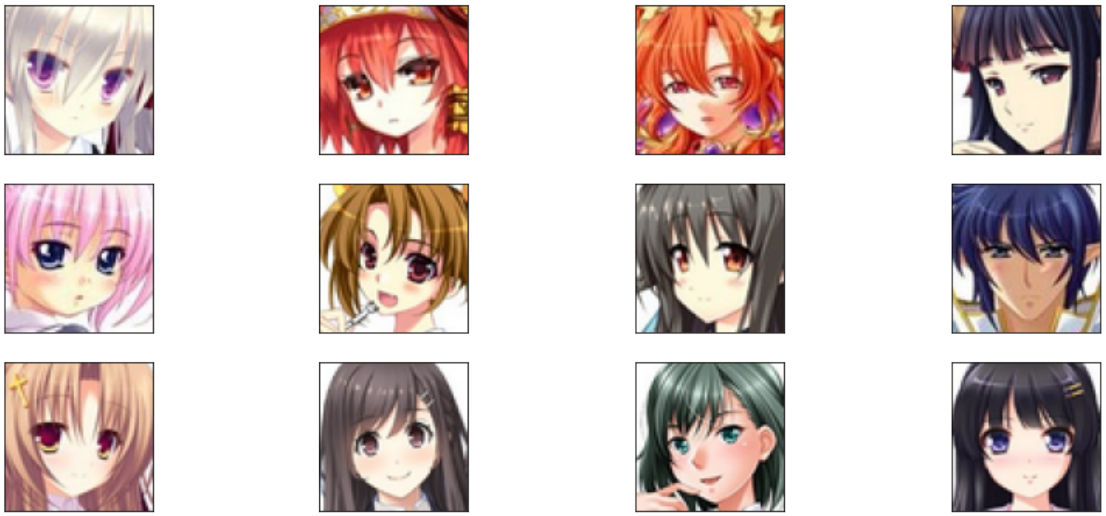
[2]:
```
# buscando as imagens e armazenando
img_size = (64, 64)
imgs = []
for f in tqdm(os.listdir('data/cropped')):
    # algumas imagens vem com tamanho = 0, pra isso foi usado o bloco try abaixo
```

```
    try:
        img = Image.open(os.path.join('data/cropped', f))
        resized_img = img.resize(img_size, Image.ANTIALIAS)
        imgs.append(resized_img)
    except Exception as e:
        pass
```

Verificação de alguns exemplos de imagens

[4]:
```
random_imgs = np.random.randint(0, len(imgs), size=(3, 4))
fig, ax = plt.subplots(3, 4, figsize=(14, 6))

for random_col in range(random_imgs.shape[1]):
    for random_row in range(random_imgs.shape[0]):
        ax[random_row, random_col].imshow(imgs[random_imgs[random_row,␣
 ↪random_col]])
        ax[random_row, random_col].get_xaxis().set_visible(False)
        ax[random_row, random_col].get_yaxis().set_visible(False)
```



Convertendo as imagens para np arrays

[5]:
```
x_all = []
for img in imgs:
    x_all.append(kp_image.img_to_array(img))

del imgs
```

[6]:
```
x_all = np.asarray(x_all)
```

[7]:
```
x_train, x_test = train_test_split(x_all, train_size=0.7, random_state=seed)
x_train = x_train/255.
x_test = x_test/255.
x_train = np.reshape(x_train, (len(x_train), 64, 64, 3))
x_test = np.reshape(x_test, (len(x_test), 64, 64, 3))
```

Criando e compilando a rede

[9]:
```
# Importa classes e funções
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, BatchNormalization, Input, Conv2D,␣
 ↪Conv2DTranspose, UpSampling2D, MaxPooling2D


input_img = Input(shape=(64, 64, 3))
encoding_dim = 32

x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')


# Add a Dense layer with a L1 activity regularizer
```

[10]:
```
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
```

```
conv2d (Conv2D)              (None, 64, 64, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 16)        9232
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 8)         1160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 16)        1168
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 64)        9280
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 3)         1731
=================================================================
Total params: 24,947
Trainable params: 24,947
Non-trainable params: 0
_________________________________________________________________
```

[11]:
```
from keras.callbacks import TensorBoard

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=128,
                verbose=0,
                validation_data=(x_test, x_test),
                callbacks=[TensorBoard(log_dir='/autoencoder')])
```

[11]: <tensorflow.python.keras.callbacks.History at 0x7feea6f90130>

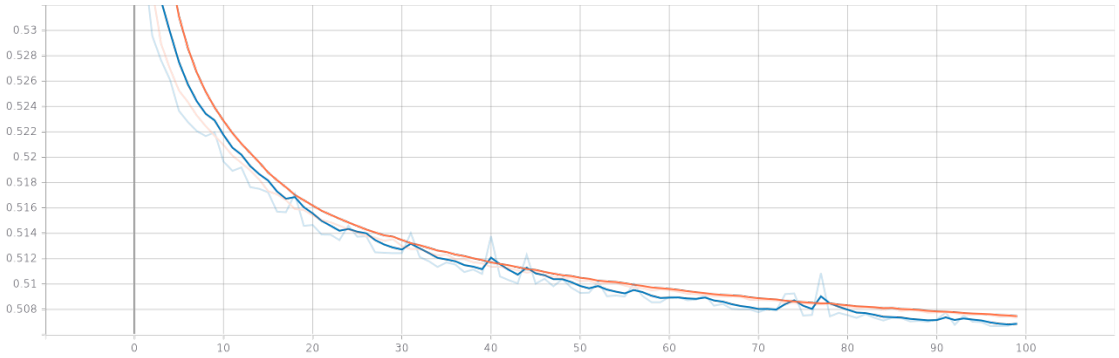


Figura retirada do TensorBoard

[16]:
```
autoencoder.save('models/autoencoder')
```

```
INFO:tensorflow:Assets written to: models/autoencoder/assets
```

Para a comparação das imagens, a métrica escolhida foi o Mean Squared Error, onde quanto menor, melhor.

[38]:
```
def mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])

    return err
```

2 Exercício 1 - Comparação entre imagens originais e transformadas

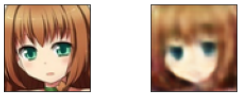
[74]:
```
#decoded_imgs = autoencoder.predict(x_test)

random_imgs = np.random.randint(0, len(x_test), size=(16))
fig, ax = plt.subplots(16, 2, figsize=(4, 25))

for random_row in range(random_imgs.shape[0]):
    ax[random_row, 0].imshow(x_test[random_imgs[random_row]].reshape(64,64,3))
    ax[random_row, 1].imshow(decoded_imgs[random_imgs[random_row]].
 ↪reshape(64,64,3))
    ax[random_row, 0].get_xaxis().set_visible(False)
    ax[random_row, 0].get_yaxis().set_visible(False)
    ax[random_row, 1].get_xaxis().set_visible(False)
    ax[random_row, 1].get_yaxis().set_visible(False)
    ax[random_row, 0].set_title("MSE= {}".
 ↪format(round(100*mse(x_test[random_imgs[random_row]].reshape(64,64,3),␣
 ↪decoded_imgs[random_imgs[random_row]].reshape(64,64,3)), 4)), x=1.35)
fig.tight_layout()
```

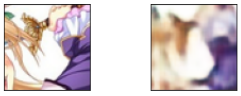
MSE= 6.8924

MSE= 4.146



MSE= 5.2016

MSE= 4.3922



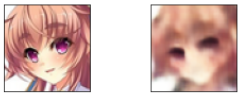
MSE= 7.107

MSE= 3.906

MSE= 6.0435

MSE= 5.9882

MSE= 4.0156



MSE= 3.9745

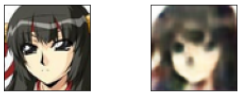
MSE= 5.9953

MSE= 4.0846

MSE= 4.3198

MSE= 6.9623

MSE= 5.8704



MSE= 6.8062

3 Exercício 2 - Superposição entre imagens transformadas

[130]:
```
low = 0
high = 1.01
step = 0.1

n = round((high - low) // step)
fig, ax = plt.subplots(8, n+1, figsize=(20,25))

for k in range(8):
    random_img = np.random.randint(0, len(x_test))
    random_target = np.random.randint(0, len(x_test))

    for i, value in enumerate(np.arange(low, high, step)):
        ax[k, i].imshow((1-value)*decoded_imgs[random_img].
 ↪reshape(64,64,3)+(value)*decoded_imgs[random_target].reshape(64,64,3))
        ax[k, i].get_xaxis().set_visible(False)
        ax[k, i].get_yaxis().set_visible(False)
    ax[k, int(n/2)].set_title("Init Img {} -> Target Img {}".format(random_img,␣
 ↪random_target), fontsize=20)
fig.tight_layout()
```

3.1 Exercício 2 Bônus - GIF da trânsição

Link para o GIF e outras imagens da aula: https://drive.google.com/drive/folders/1NMKm-bMNGT

[144]:
```
from celluloid import Camera
low = 0
high = 1.01
```

```
step = 0.01

n = round((high - low) // step)
fig = plt.figure(figsize=(16, 16))
cam = Camera(fig)

random_img = np.random.randint(0, len(x_test))
random_target = np.random.randint(0, len(x_test))
ax = plt.gca()
for i in range(30):
    plt.imshow(x_test[random_img].reshape(64,64,3))
    cam.snap()
for i, value in enumerate(np.arange(low, high, step)):
    plt.imshow((1-value)*decoded_imgs[random_img].
 ↪reshape(64,64,3)+(value)*decoded_imgs[random_target].reshape(64,64,3))
    ax.axes.xaxis.set_visible(False)
    ax.axes.yaxis.set_visible(False)
    cam.snap()
for i in range(30):
    plt.imshow(x_test[random_target].reshape(64,64,3))
    cam.snap()
anim = cam.animate(interval=70)

anim.save('transition.gif');
```