

Lab 7 - BCC406

REDES NEURAIIS E APRENDIZAGEM EM PROFUNDIDADE

Modelos Generativos

Prof. Eduardo e Prof. Pedro

Objetivos:

- Parte I : Compressão com AE
- Parte II : Detecção de anomalias
- Parte III: Redes Generativas Adversariais

Data da entrega : 21/10

- Complete o código (marcado com `ToDo`) e quando requisitado, escreva textos diretamente nos notebooks. Onde tiver *None*, substitua pelo seu código.
- Execute todo notebook e salve tudo em um PDF **nomeado** como "NomeSobrenome-Lab.pdf"
- Envie o PDF via google [FORM](#)

Este notebook é baseado em tensorflow e Keras.

▼ Parte I: Autoencoder para redução de dimensionalidade (30pt)

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4 import tensorflow as tf
5
6 from sklearn.metrics import accuracy_score, precision_score, recall_score
7 from sklearn.model_selection import train_test_split
8 from tensorflow.keras import layers, losses
9 from tensorflow.keras.datasets import fashion_mnist
10 from tensorflow.keras.models import Model
```

Carrega dataset Fashion MNIST dataset. Cada imagem tem resolução 28x28 pixels.

```

1 (x_train, _), (x_test, _) = fashion_mnist.load_data()
2
3 x_train = x_train.astype('float32') / 255.
4 x_test = x_test.astype('float32') / 255.
5
6 print (x_train.shape)
7 print (x_test.shape)

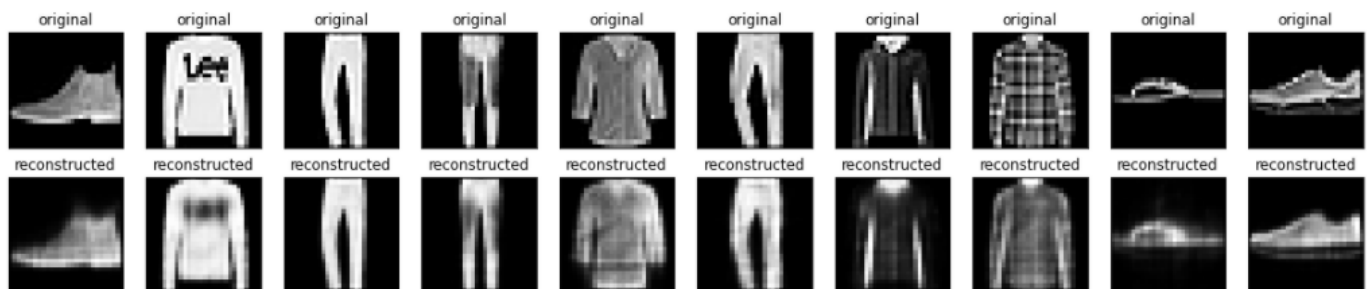
```

```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
29515/29515 ————— 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
26421880/26421880 ————— 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
5148/5148 ————— 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
4422102/4422102 ————— 0s 0us/step
(60000, 28, 28)
(10000, 28, 28)

```

✓ Exemplo de classes



Abaixo exemplo de implementação de autoencoder apenas com camadas densas. O encoder, comprime as imagens em 4 dimensões (latent_dim), e o decoder reconstrói a imagem a partir do vetor latente.

O exemplo abaixo usa a [Keras Model Subclassing API](#).

```

1 latent_dim = 4
2
3 class Autoencoder(Model):
4     def __init__(self, latent_dim):
5         super(Autoencoder, self).__init__()
6         self.latent_dim = latent_dim
7         self.encoder = tf.keras.Sequential([
8             layers.Flatten(),
9             layers.Dense(latent_dim, activation='relu'),
10        ])

```

```
11 self.decoder = tf.keras.Sequential([
12     layers.Dense(784, activation='sigmoid'),
13     layers.Reshape((28, 28))
14 ])
15
16 def call(self, x):
17     encoded = self.encoder(x)
18     decoded = self.decoder(encoded)
19     return decoded
20
21 autoencoder = Autoencoder(latent_dim)
```


```
1 autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

```
1 autoencoder.summary()
```

```
1 autoencoder.fit(x_train, x_train,
2                 epochs=10,
3                 shuffle=True,
4                 validation_data=(x_test, x_test))
```

```
Epoch 1/10
1875/1875 ————— 8s 3ms/step - loss: 0.0702 - val_loss: 0.0377
Epoch 2/10
1875/1875 ————— 5s 3ms/step - loss: 0.0365 - val_loss: 0.0340
Epoch 3/10
1875/1875 ————— 4s 2ms/step - loss: 0.0337 - val_loss: 0.0331
Epoch 4/10
1875/1875 ————— 6s 3ms/step - loss: 0.0331 - val_loss: 0.0328
Epoch 5/10
1875/1875 ————— 4s 2ms/step - loss: 0.0329 - val_loss: 0.0327
Epoch 6/10
1875/1875 ————— 4s 2ms/step - loss: 0.0328 - val_loss: 0.0327
Epoch 7/10
1875/1875 ————— 5s 3ms/step - loss: 0.0327 - val_loss: 0.0326
Epoch 8/10
1875/1875 ————— 4s 2ms/step - loss: 0.0327 - val_loss: 0.0326
Epoch 9/10
1875/1875 ————— 5s 3ms/step - loss: 0.0328 - val_loss: 0.0326
```

Epoch 10/10

1875/1875  5s 3ms/step - loss: 0.0325 - val_loss: 0.0326

<keras.src.callbacks.history.History at 0x7a2f3ffeabd0>

Treine o modelo e veja os resultados da re-construção.

```
1 encoded_imgs = autoencoder.encoder(x_test).numpy()
2 decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()

1 n = 10
2 plt.figure(figsize=(20, 4))
3 for i in range(n):
4     # display original
5     ax = plt.subplot(2, n, i + 1)
6     plt.imshow(x_test[i])
7     plt.title("original")
8     plt.gray()
9     ax.get_xaxis().set_visible(False)
10    ax.get_yaxis().set_visible(False)
11
12    # display reconstruction
13    ax = plt.subplot(2, n, i + 1 + n)
14    plt.imshow(decoded_imgs[i])
15    plt.title("reconstructed")
16    plt.gray()
17    ax.get_xaxis().set_visible(False)
18    ax.get_yaxis().set_visible(False)
19 plt.show()
```

✓ ToDo : Testes (15pt)

```
1 def plot_results(autoencoder):
2     encoded_imgs = autoencoder.encoder(x_test).numpy()
3     decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
4
5     n = 10
6     plt.figure(figsize=(20, 4))
7     for i in range(n):
8         # display original
9         ax = plt.subplot(2, n, i + 1)
10        plt.imshow(x_test[i])
11        plt.title("original")
12        plt.gray()
13        ax.get_xaxis().set_visible(False)
14        ax.get_yaxis().set_visible(False)
15
16        # display reconstruction
17        ax = plt.subplot(2, n, i + 1 + n)
18        plt.imshow(decoded_imgs[i])
19        plt.title("reconstructed")
20        plt.gray()
21        ax.get_xaxis().set_visible(False)
22        ax.get_yaxis().set_visible(False)
23    plt.show()
24
25 def compile_and_train_model(latent_dim):
26     autoencoder = Autoencoder(latent_dim)
27     autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
28     autoencoder.fit(x_train, x_train, epochs=10, shuffle=True, validation_data=(x_test
29     plot_results(autoencoder)
```

Faça testes com vetor latente de dimensões 2, 8, 16 e 64.

```
1 compile_and_train_model(2)
```

```
1 compile_and_train_model(8)
```

```
1 compile_and_train_model(16)
```

```
1 compile_and_train_model(64)
```

✓ ToDo : Responda (15pt)

Escreva suas conclusões sobre os testes executados

É possível perceber correlação entre o aumento do tamanho da camada latente e a qualidade final das imagens reconstruídas.

Tomando como exemplo a segunda imagem (blusa de manga comprida), é notável a melhora na tentativa de representação da estampa da camisa.

Tal melhora também pode ser percebida nas calças em que uma das pernas está levemente curvada, onde, com mais neurônios, a rede é capaz de representar melhor esta movimentação.

Na penúltima imagem, é possível perceber uma sombra que se assemelha a outra roupa, todavia ela vai se dissipando e a imagem objetivo fica mais nítida.

✓ Parte II: Detecção de anomalias (30pt)

Intro

Neste exemplo, você vai detectar anomalias em sinais de eletrocardiograma (ECG). Para tal

Neste exemplo, você vai detectar anomalias em sinais de eletrocardiograma (ECG). Para tal, treine um autoencoder no dataset [ECG5000 dataset](#). Este dataset contém 5000 batimentos de ECG (<https://en.wikipedia.org/wiki/Electrocardiography>), cada um com 140 amostras (pontos) na curva. Cada instância da base de dados (um batimento) foi rotulado como zero (0) ou um (1). A classe zero corresponde a um batimento anormal e a classe um a um batimento de classe normal. Queremos identificar os anormais.

Para detectar anomalias usando um autoencoder você deve treinar um autoencoder apenas em batimentos normais. Ele vai aprender a re-construir os batimentos saudáveis. A hipótese é que os batimentos anormais vão divergir no padrão, quando compararmos a entrada com a re-construção.

✓ Carrega base de ECG

Base de dados detalhada no site: timeseriesclassification.com.

```
1 # Download the dataset
2 dataframe = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/e
3 raw_data = dataframe.values
4 dataframe.head()
```

```
1 # The last element contains the labels
2 labels = raw_data[:, -1]
3
4 # The other data points are the electrocardiogram data
5 data = raw_data[:, 0:-1]
6
7 train_data, test_data, train_labels, test_labels = train_test_split(
8     data, labels, test_size=0.2, random_state=21
9 )
```

```
..      .      .      .      .      .
```


Normaliza entre $[0,1]$.

```
1 min_val = tf.reduce_min(train_data)
2 max_val = tf.reduce_max(train_data)
3
4 train_data = (train_data - min_val) / (max_val - min_val)
5 test_data = (test_data - min_val) / (max_val - min_val)
6
7 train_data = tf.cast(train_data, tf.float32)
8 test_data = tf.cast(test_data, tf.float32)
```

Vamos separar os batimentos normais (label 1) para treinar o Autoencoder.

```
1 train_labels = train_labels.astype(bool)
2 test_labels = test_labels.astype(bool)
3
4 normal_train_data = train_data[train_labels]
5 normal_test_data = test_data[test_labels]
6
7 anomalous_train_data = train_data[~train_labels]
8 anomalous_test_data = test_data[~test_labels]
```

Plote um batimento normal.

```
1 plt.grid()
2 plt.plot(np.arange(140), normal_train_data[0])
3 plt.title("A Normal ECG")
4 plt.show()
```

Plote um batimento anômalo.

```
1 plt.grid()
2 plt.plot(np.arange(140), anomalous_train_data[0])
3 plt.title("An Anomalous ECG")
4 plt.show()
```

✓ ToDo : Construção de um modelo (30pt)

Construa um modelo. Primeiramente tente construir apenas com camadas densas. Depois, tente construir um modelo com camadas de convolução de uma dimensão (Lembre-se que um sinal de ECG é uma série temporal de uma dimensão). [Conv1D](#)

```
1
2 class AnomalyDetector(Model):
3     def __init__(self):
```

```

3  def __init__(self):
4      super(AnomalyDetector, self).__init__()
5
6      self.encoder = tf.keras.Sequential([
7          layers.Input(shape=(140,)),
8          layers.Dense(64, activation="relu"),
9          layers.Dense(64, activation="relu"),
10     ])
11     self.decoder = tf.keras.Sequential([
12         layers.Dense(140, activation='sigmoid'),
13     ])
14
15     def call(self, x):
16         encoded = self.encoder(x)
17         decoded = self.decoder(encoded)
18         return decoded
19
20 autoencoder = AnomalyDetector()

1 autoencoder.compile(optimizer='adam', loss='mae')

```

Depois de treinar com os batimentos normais, avalie com os anormais.

```

1 history = autoencoder.fit(normal_train_data, normal_train_data,
2     epochs=20,
3     batch_size=512,
4     validation_data=(test_data, test_data),
5     shuffle=True)

```

```

Epoch 1/20
5/5 ————— 3s 347ms/step - loss: 0.0639 - val_loss: 0.0546
Epoch 2/20
5/5 ————— 0s 24ms/step - loss: 0.0525 - val_loss: 0.0504
Epoch 3/20
5/5 ————— 0s 30ms/step - loss: 0.0459 - val_loss: 0.0475
Epoch 4/20
5/5 ————— 0s 21ms/step - loss: 0.0400 - val_loss: 0.0440
Epoch 5/20
5/5 ————— 0s 30ms/step - loss: 0.0353 - val_loss: 0.0418
Epoch 6/20
5/5 ————— 0s 29ms/step - loss: 0.0313 - val_loss: 0.0401
Epoch 7/20
5/5 ————— 0s 30ms/step - loss: 0.0286 - val_loss: 0.0385
Epoch 8/20
5/5 ————— 0s 31ms/step - loss: 0.0266 - val_loss: 0.0370
Epoch 9/20
5/5 ————— 0s 15ms/step - loss: 0.0250 - val_loss: 0.0358
Epoch 10/20
5/5 ————— 0s 15ms/step - loss: 0.0237 - val_loss: 0.0348
Epoch 11/20
5/5 ————— 0s 15ms/step - loss: 0.0224 - val_loss: 0.0341

```

```

Epoch 12/20
5/5 ----- 0s 14ms/step - loss: 0.0214 - val_loss: 0.0337
Epoch 13/20
5/5 ----- 0s 14ms/step - loss: 0.0203 - val_loss: 0.0333
Epoch 14/20
5/5 ----- 0s 15ms/step - loss: 0.0195 - val_loss: 0.0330
Epoch 15/20
5/5 ----- 0s 25ms/step - loss: 0.0193 - val_loss: 0.0325
Epoch 16/20
5/5 ----- 0s 15ms/step - loss: 0.0191 - val_loss: 0.0320
Epoch 17/20
5/5 ----- 0s 16ms/step - loss: 0.0185 - val_loss: 0.0315
Epoch 18/20
5/5 ----- 0s 14ms/step - loss: 0.0184 - val_loss: 0.0310
Epoch 19/20
5/5 ----- 0s 14ms/step - loss: 0.0182 - val_loss: 0.0308
Epoch 20/20
5/5 ----- 0s 14ms/step - loss: 0.0178 - val_loss: 0.0300

```

```

1 plt.plot(history.history["loss"], label="Training Loss")
2 plt.plot(history.history["val_loss"], label="Validation Loss")
3 plt.legend()

```

Você vai considerar um batimento como anômalo se ele divergir mais que um desvio padrão das amostras normais. Primeiro, vamos plotar um batimento normal a partir da base de treino e sua reconstrução. Assim, poderemos calcular o erro de re-construção.

```
1 encoded_data = autoencoder.encoder(normal_test_data).numpy()
2 decoded_data = autoencoder.decoder(encoded_data).numpy()
3
4 plt.plot(normal_test_data[0], 'b')
5 plt.plot(decoded_data[0], 'r')
6 plt.fill_between(np.arange(140), decoded_data[0], normal_test_data[0], color='lightco
7 plt.legend(labels=["Input", "Reconstruction", "Error"])
8 plt.show()
```

Vamos fazer o mesmo para um batimento anômalo.

```
1 encoded_data = autoencoder.encoder(anomalous_test_data).numpy()
2 decoded_data = autoencoder.decoder(encoded_data).numpy()
3
4 plt.plot(anomalous_test_data[0], 'b')
5 plt.plot(decoded_data[0], 'r')
6 plt.fill_between(np.arange(140), decoded_data[0], anomalous_test_data[0], color='ligh
7 plt.legend(labels=["Input", "Reconstruction", "Error"])
8 plt.show()
```

▼ Detectando as anomalias

Vamos detectar as anomalias se o erro de reconstrução for maior que um limiar. Aqui, vamos calcular o erro médio para os exemplos normais do treino e depois, classificar os anormais do teste, que tenha erro de reconstrução maior que um desvio padrão.

Plota erro de reconstrução de batimentos normais do treino

```
1 reconstructions = autoencoder.predict(normal_train_data)
2 train_loss = tf.keras.losses.mae(reconstructions, normal_train_data)
3
4 plt.hist(train_loss[None,:], bins=50)
5 plt.xlabel("Train loss")
6 plt.ylabel("No of examples")
7 plt.show()
```

Escolha do limiar.

```
1 threshold = np.mean(train_loss) + np.std(train_loss)
2 print("Threshold: ", threshold)
```

```
Threshold:  0.027278356
```

```
1 reconstructions = autoencoder.predict(anomalous_test_data)
2 test_loss = tf.keras.losses.mae(reconstructions, anomalous_test_data)
3
4 plt.hist(test_loss[None, :], bins=50)
5 plt.xlabel("Test loss")
6 plt.ylabel("No of examples")
7 plt.show()
```

Classificação.

```
1 def predict(model, data, threshold):
2     reconstructions = model(data)
3     loss = tf.keras.losses.mae(reconstructions, data)
4     return tf.math.less(loss, threshold)
5
6 def print_stats(predictions, labels):
7     print("Accuracy = {}".format(accuracy_score(labels, predictions)))
8     print("Precision = {}".format(precision_score(labels, predictions)))
9     print("Recall = {}".format(recall_score(labels, predictions)))
```

Calcule a acurácia para os dois modelos (com camadas densas e convolucionais)

```
1 preds = predict(autoencoder, test_data, threshold)
2 print_stats(preds, test_labels)
```

```
Accuracy = 0.943
Precision = 0.9941060903732809
Recall = 0.9035714285714286
```

✓ Parte III: Redes Generativas Adversariais (40pt)

Leia o tutorial sobre a pix2pix em [Tensorflow Tutorials](#). O pix2pix foi apresentado em [Image-to-image translation with conditional adversarial networks by Isola et al. \(2017\)](#) e se trata de uma rede generativa adversarial condicional para geração de fachadas de prédios condicionada a uma máscara representando a arquitetura. baixe o notebook do tutorial, estude e treine a GAN. Após o treinamento, construa você mesmo 3 máscaras (usando algum software de desenho) e faça uma inferência com a rede. Anexe no notebook a máscara e sua respectiva saída.

✓ ToDo : Fachadas de prédios (40pt)

```
1 # ToDo : Criar 3 máscaras e gerar 3 saídas com a pix2pix para o problema de fachadas
```

Baseado nos exemplos do tensorflow [tutorials](#)

