Lab 6 - PCC177/BCC406

REDES NEURAIS 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 : XX/XX

- 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)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
```

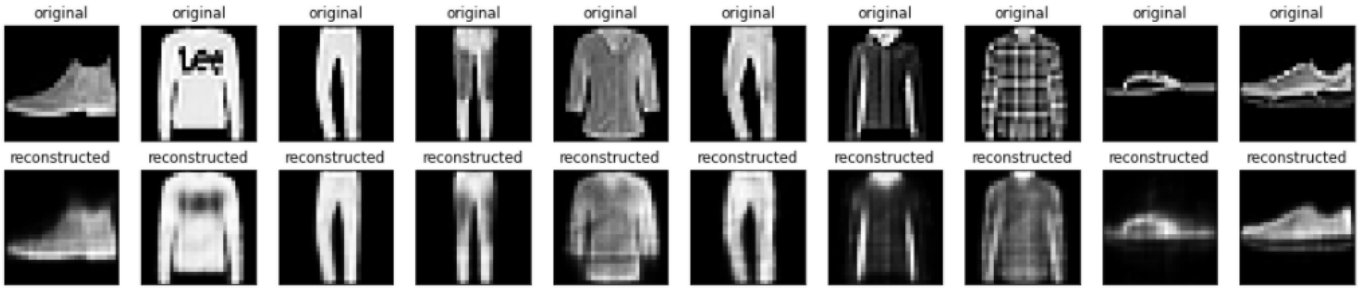

Carrega dataset Fashon MNIST dataset. Cada imagem tem resolução 28x28 pixels.

```
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
```

```
x_test = x_test.astype('float32') / 255.

print (x_train.shape)
print (x_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

Exemplo de classes



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

O exemplo abaixo usa a Keras Model Subclassing API.

```
latent_dim = 4

class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
```



```
autoencoder = Autoencoder(latent_dim)


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


autoencoder.fit(x_train, x_train,
```

```
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

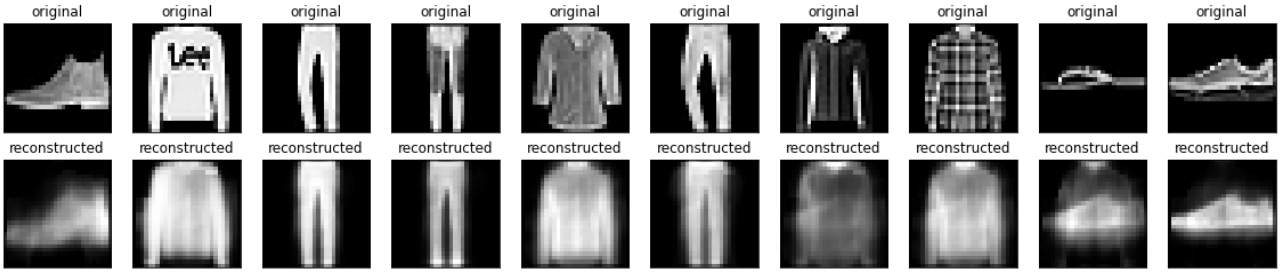
```
Epoch 1/10
1875/1875 [==============================] - 7s 2ms/step - loss: 0.0504 - val_loss:
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0359 - val_loss:
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0337 - val_loss:
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0331 - val_loss:
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0328 - val_loss:
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0327 - val_loss:
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0326 - val_loss:
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0326 - val_loss:
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0325 - val_loss:
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0325 - val_loss:
<keras.callbacks.History at 0x7f86a01c4990>
```

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

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


n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
```



```
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```



ToDo : Testes (15pt)

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

Dim=2

```
latent_dim = 2

class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Autoencoder(latent_dim)


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

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

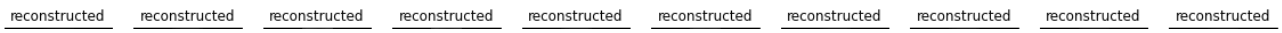
```
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0688 - val_loss:
Epoch 2/10
```

```
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0548 - val_loss:
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0486 - val_loss:
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0454 - val_loss:
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0446 - val_loss:
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0444 - val_loss:
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0443 - val_loss:
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0442 - val_loss:
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0442 - val_loss:
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0442 - val_loss:
<keras.callbacks.History at 0x7f863612ac50>
```

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


n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

Dim=8



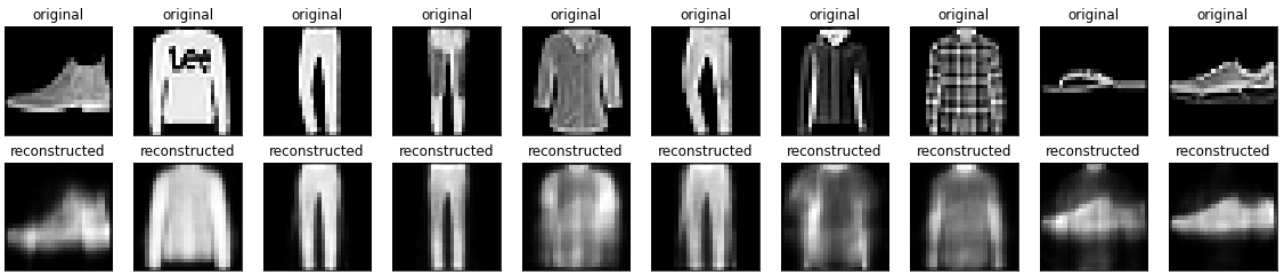

```
latent_dim = 8

class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Autoencoder(latent_dim)


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


autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0407 - val_loss:
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0251 - val_loss:
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0237 - val_loss:
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0234 - val_loss:
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0232 - val_loss:
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0231 - val_loss:
```



```
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0230 - val_loss:
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0230 - val_loss:
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0230 - val_loss:
<keras.callbacks.History at 0x7f86204ff350>
```

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


n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```



Dim=16

```
latent_dim = 16

class Autoencoder(Model):
```



```
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
```

```
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Autoencoder(latent_dim)

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


autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

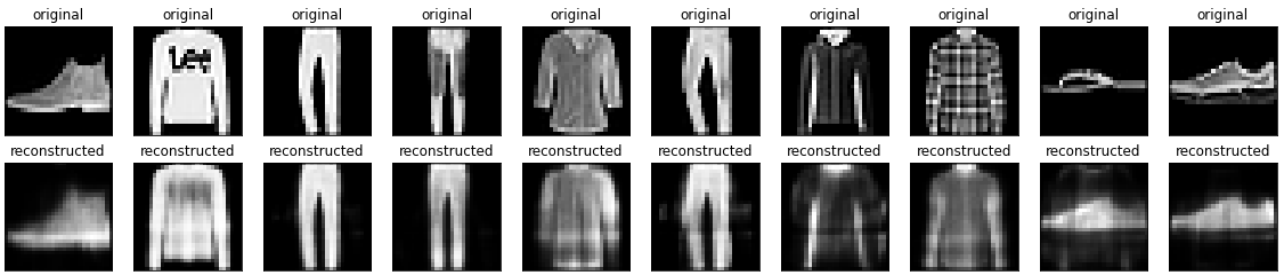
```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0346 - val_loss:
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0199 - val_loss:
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0182 - val_loss:
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0179 - val_loss:
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0177 - val_loss:
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0176 - val_loss:
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0175 - val_loss:
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0175 - val_loss:
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0174 - val_loss:
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0174 - val_loss:
<keras.callbacks.History at 0x7f86203ce190>
```

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


n = 10
plt.figure(figsize=(20, 4))
```



```
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
```

```
  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```



Dim=64

```
 latent_dim =64

class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded
```

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

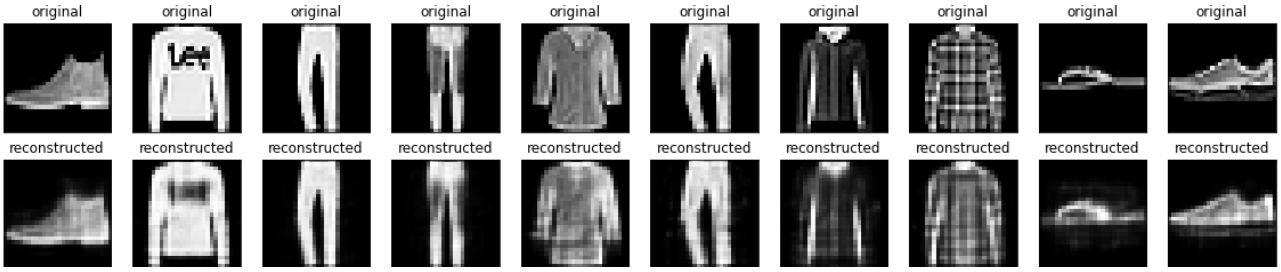
```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())


autoencoder.fit(x_train, x_train,
                epochs=10,
```

```
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0237 - val_loss:
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0116 - val_loss:
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0102 - val_loss:
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0096 - val_loss:
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0092 - val_loss:
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0091 - val_loss:
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0090 - val_loss:
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0089 - val_loss:
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0088 - val_loss:
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0088 - val_loss:
<keras.callbacks.History at 0x7f8636304d10>
```

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

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
```



ToDo : Responda (15pt)

Escreva suas conclusões sobre os testes executados:

Com base nos testes executados, podemos concluir que com o aumento do vetor latente ocorre uma melhora na resolução da imagem e tambem uma interferencia menor entre as imagens geradas.

Parte II: Detecção de anomalias (30pt)

Intro

Neste exemplo, você vai detectar anomalis em sinaos 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 hipóteses é que os batimentos anormais vão divergir no padrão, quando compararmos a entrada com a re-construção.

Carrega base de ECG

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

```
# Download the dataset
dataframe = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.cs
raw_data = dataframe.values
dataframe.head()
```

	0	1	2	3	4	5	6	7
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423

5 rows × 141 columns

```
# The last element contains the labels
labels = raw_data[:, -1]

# The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]

train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=21
)
```

Normaliza entre `[0,1]`.

```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)

train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)
```

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

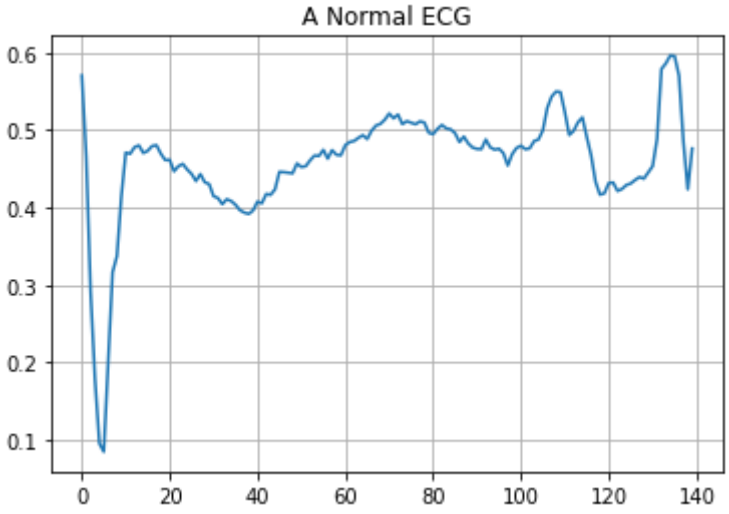
```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

normal_train_data = train_data[train_labels]
normal_test_data = test_data[test_labels]
```

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

```
anomalous_test_data = test_data[~test_labels]
```

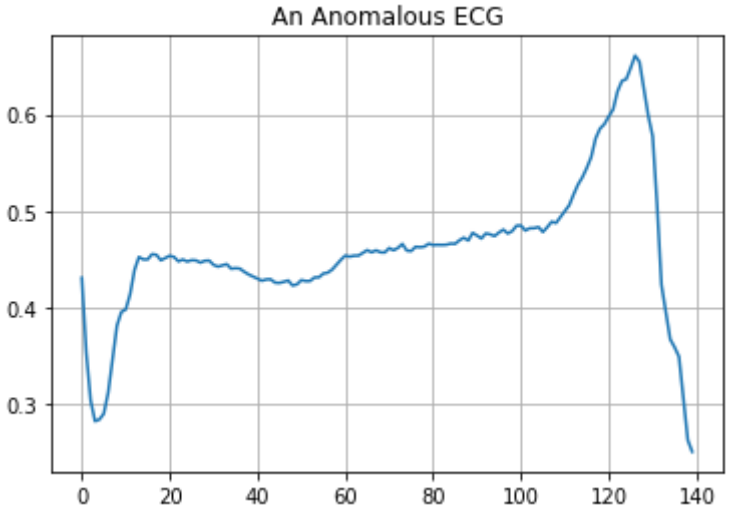
Plote um batimento normal.

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



Plote um batimento anômalo.

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



ToDo : Construção de um modelo (30pt)

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

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

```
latent_dim = 8
```

```
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(140, activation='sigmoid'),
      #layers.Reshape((140,0))
    ])
    #self.encoder = tf.keras.Sequential([
      # Todo: crie o encoder, com um gargalo

    #self.decoder = tf.keras.Sequential([
      # Todo: crie as camadas do decoder

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = AnomalyDetector()


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

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

```
history = autoencoder.fit(normal_train_data, normal_train_data,
          epochs=20,
          batch_size=512,
          validation_data=(test_data, test_data),
          shuffle=True)
```

```
Epoch 1/20
5/5 [==============================] - 1s 29ms/step - loss: 0.0571 - val_loss: 0.052
Epoch 2/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0547 - val_loss: 0.0510
Epoch 3/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0524 - val_loss: 0.049
Epoch 4/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0500 - val_loss: 0.048
Epoch 5/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0474 - val_loss: 0.0469
```

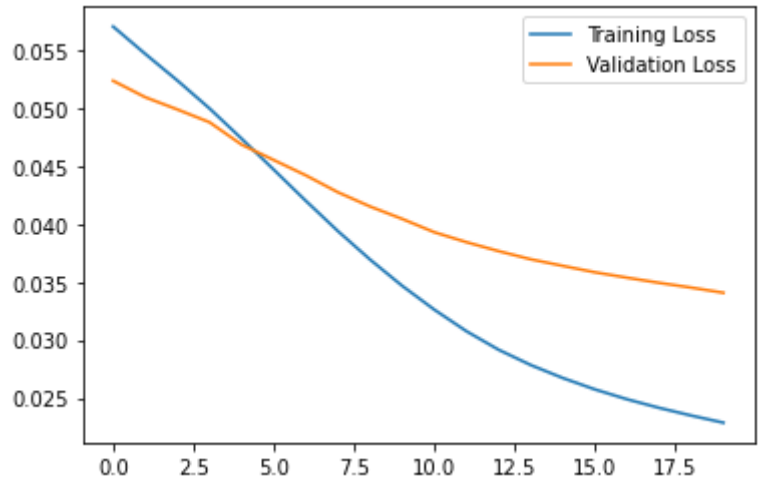


```
5/5 [==============================] - 0s 9ms/step - loss: 0.0420 - val_loss: 0.0442
Epoch 8/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0394 - val_loss: 0.0428
Epoch 9/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0369 - val_loss: 0.0415
Epoch 10/20
```

```
5/5 [==============================] - 0s 8ms/step - loss: 0.0347 - val_loss: 0.0405
Epoch 11/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0326 - val_loss: 0.0393
Epoch 12/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0308 - val_loss: 0.038
Epoch 13/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0292 - val_loss: 0.0377
Epoch 14/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0278 - val_loss: 0.0370
Epoch 15/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0267 - val_loss: 0.036
Epoch 16/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0257 - val_loss: 0.0358
Epoch 17/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0249 - val_loss: 0.0354
Epoch 18/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0241 - val_loss: 0.0349
Epoch 19/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0235 - val_loss: 0.034
Epoch 20/20
5/5 [==============================] - 0s 8ms/step - loss: 0.0229 - val_loss: 0.0341
```

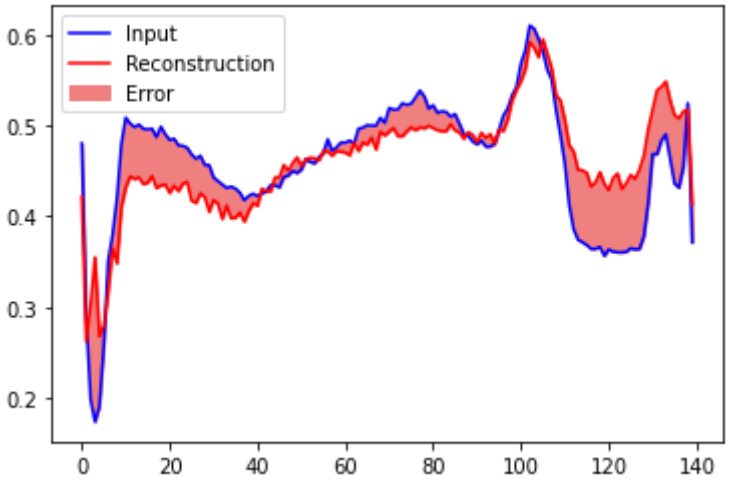
```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f85fcdda990>
```



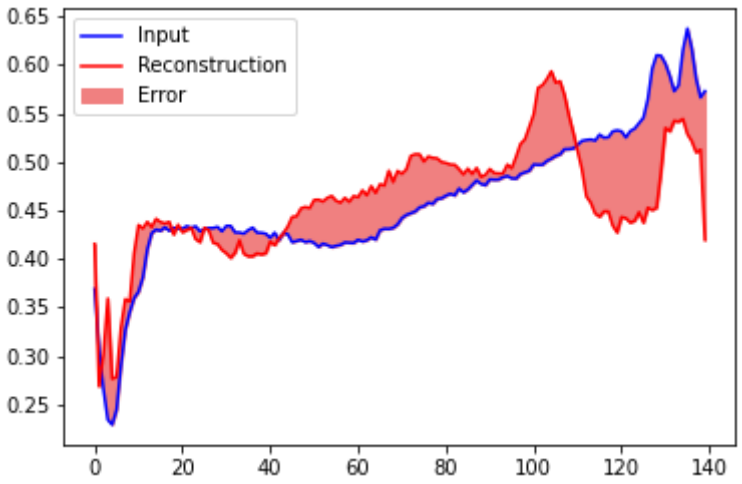
Você vai considerar um batimento como anômalo se ele divergir masi que um desvio padrão das amostras normais. Primeiro, vamos plotar um batimento normal a partir da base de treino e sua reconstrução. Assim, poderemos calcular o erro de re-construção.


```
plt.plot(normal_test_data[0], 'b')
plt.plot(decoded_data[0], 'r')
plt.fill_between(np.arange(140), decoded_data[0], normal_test_data[0], color='lightcoral')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



Vamos fazer o mesmo para um batimento anômalo.

```
encoded_data = autoencoder.encoder(anomalous_test_data).numpy()
decoded_data = autoencoder.decoder(encoded_data).numpy()

plt.plot(anomalous_test_data[0], 'b')
plt.plot(decoded_data[0], 'r')
plt.fill_between(np.arange(140), decoded_data[0], anomalous_test_data[0], color='lightcora
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



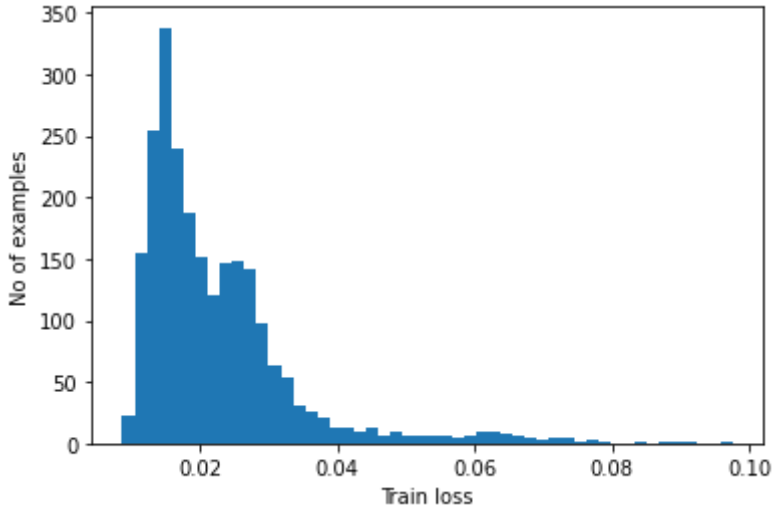
Detectando as anomalias

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

calcular o erro médio para os exemp;los 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

```
reconstructions = autoencoder.predict(normal_train_data)
train_loss = tf.keras.losses.mae(reconstructions, normal_train_data)

plt.hist(train_loss[None,:], bins=50)
plt.xlabel("Train loss")
plt.ylabel("No of examples")
plt.show()
```



Escolha do limiar.

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

```
Threshold:  0.034112945
```

```
reconstructions = autoencoder.predict(anomalous_test_data)
test_loss = tf.keras.losses.mae(reconstructions, anomalous_test_data)

plt.hist(test_loss[None, :], bins=50)
plt.xlabel("Test loss")
plt.ylabel("No of examples")
plt.show()
```

Classificação.


```
def predict(model, data, threshold):
  reconstructions = model(data)
  loss = tf.keras.losses.mae(reconstructions, data)
  return tf.math.less(loss, threshold)

def print_stats(predictions, labels):
  print("Accuracy = {}".format(accuracy_score(labels, predictions)))
  print("Precision = {}".format(precision_score(labels, predictions)))
  print("Recall = {}".format(recall_score(labels, predictions)))
```

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

```
preds = predict(autoencoder, test_data, threshold)
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 Tensofrflow 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 reprtesentando a arquitetura. baixe o noterbook 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)


```
import pathlib
import time
import datetime

from matplotlib import pyplot as plt
from IPython import display
```

```
dataset_name = "facades"


_URL = f'http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/{dataset_name}.tar.gz'

path_to_zip = tf.keras.utils.get_file(
    fname=f"{dataset_name}.tar.gz",
    origin=_URL,
    extract=True)

path_to_zip  = pathlib.Path(path_to_zip)

PATH = path_to_zip.parent/dataset_name
```

```
Downloading data from http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/facades.ta
30171136/30168306 [==============================] - 19s 1us/step
30179328/30168306 [==============================] - 19s 1us/step
```

```
list(PATH.parent.iterdir())
```

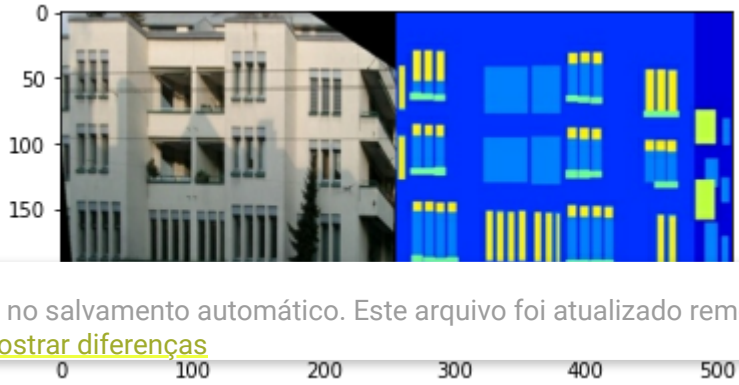
```
[PosixPath('/root/.keras/datasets/facades'),
 PosixPath('/root/.keras/datasets/fashion-mnist'),
 PosixPath('/root/.keras/datasets/facades.tar.gz')]
```

```
sample_image = tf.io.read_file(str(PATH / 'train/1.jpg'))
sample_image = tf.io.decode_jpeg(sample_image)
print(sample_image.shape)
```

```
(256, 512, 3)
```

```
plt.figure()
plt.imshow(sample_image)
```

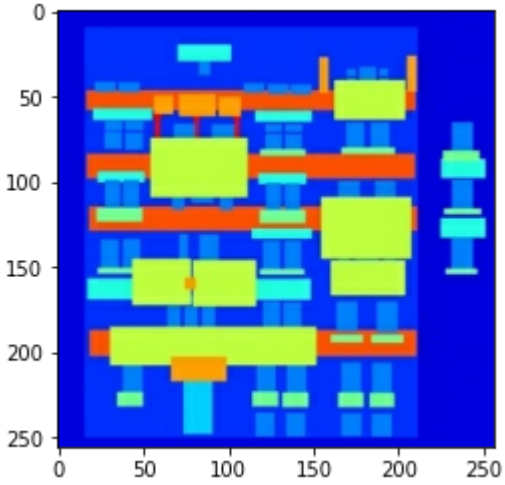
```
<matplotlib.image.AxesImage at 0x7f85fc1cfa10>
```

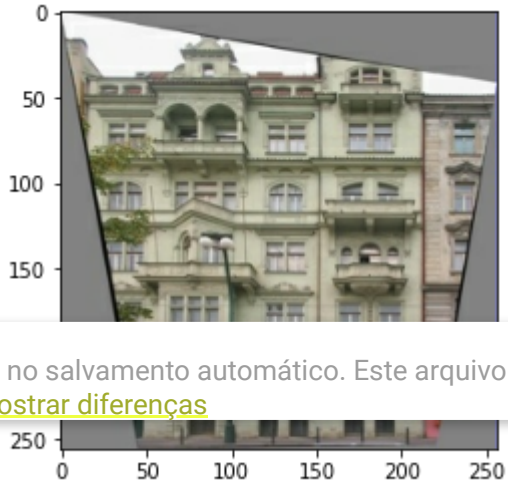



```
def load(image_file):
  # Read and decode an image file to a uint8 tensor
  image = tf.io.read_file(image_file)
```

```
  image = tf.io.decode_jpeg(image)

  # Split each image tensor into two tensors:
  # - one with a real building facade image
  # - one with an architecture label image
  w = tf.shape(image)[1]
  w = w // 2
  input_image = image[:, w:, :]
  real_image = image[:, :w, :]

  # Convert both images to float32 tensors
  input_image = tf.cast(input_image, tf.float32)
  real_image = tf.cast(real_image, tf.float32)

  return input_image, real_image


inp, re = load(str(PATH / 'train/100.jpg'))
# Casting to int for matplotlib to display the images
plt.figure()
plt.imshow(inp / 255.0)
plt.figure()
plt.imshow(re / 255.0)
```

```
<matplotlib.image.AxesImage at 0x7f85fcc70710>
```





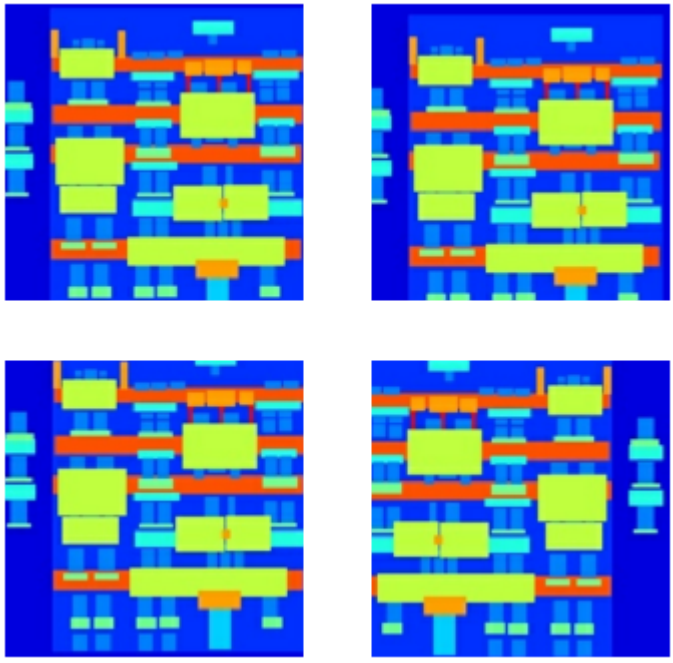
Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

```
# The facade training set consist of 400 images
BUFFER_SIZE = 400
```

```
# The batch size of 1 produced better results for the U-Net in the original pix2pix experi
BATCH_SIZE = 1
# Each image is 256x256 in size
IMG_WIDTH = 256
IMG_HEIGHT = 256

def resize(input_image, real_image, height, width):
  input_image = tf.image.resize(input_image, [height, width],
                                method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
  real_image = tf.image.resize(real_image, [height, width],
                               method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)

  return input_image, real_image


def random_crop(input_image, real_image):
  stacked_image = tf.stack([input_image, real_image], axis=0)
  cropped_image = tf.image.random_crop(
      stacked_image, size=[2, IMG_HEIGHT, IMG_WIDTH, 3])

  return cropped_image[0], cropped_image[1]


# Normalizing the images to [-1, 1]
def normalize(input_image, real_image):
  input_image = (input_image / 127.5) - 1
  real_image = (real_image / 127.5) - 1

  return input_image, real_image


@tf.function()
def random_jitter(input_image, real_image):
  # Resizing to 286x286
  input_image, real_image = resize(input_image, real_image, 286, 286)

  # Random cropping back to 256x256
  input_image, real_image = random_crop(input_image, real_image)

  if tf.random.uniform(()) > 0.5:
    # Random mirroring
    input_image = tf.image.flip_left_right(input_image)
    real_image = tf.image.flip_left_right(real_image)

  return input_image, real_image
```



```
  plt.subplot(2, 2, i + 1)
  plt.imshow(rj_inp / 255.0)
  plt.axis('off')
plt.show()
```



```
def load_image_train(image_file):
  input_image, real_image = load(image_file)
  input_image, real_image = random_jitter(input_image, real_image)
  input_image, real_image = normalize(input_image, real_image)

  return input_image, real_image


def load_image_test(image_file):
  input_image, real_image = load(image_file)
  input_image, real_image = resize(input_image, real_image,
                                   IMG_HEIGHT, IMG_WIDTH)
  input_image, real_image = normalize(input_image, real_image)

  return input_image, real_image


train_dataset = tf.data.Dataset.list_files(str(PATH / 'train/*.jpg'))
train_dataset = train_dataset.map(load_image_train,
                                  num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.batch(BATCH_SIZE)


try:
  test_dataset = tf.data.Dataset.list_files(str(PATH / 'test/*.jpg'))
except tf.errors.InvalidArgumentError:
  test_dataset = tf.data.Dataset.list_files(str(PATH / 'val/*.jpg'))
test_dataset = test_dataset.map(load_image_test)
```

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

```
OUTPUT_CHANNELS = 3


def downsample(filters, size, apply_batchnorm=True):
  initializer = tf.random_normal_initializer(0., 0.02)
```

```
  result = tf.keras.Sequential()
  result.add(
      tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',
                             kernel_initializer=initializer, use_bias=False))

  if apply_batchnorm:
    result.add(tf.keras.layers.BatchNormalization())

  result.add(tf.keras.layers.LeakyReLU())

  return result


down_model = downsample(3, 4)
down_result = down_model(tf.expand_dims(inp, 0))
print (down_result.shape)
```

```
(1, 128, 128, 3)
```

```
def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
    tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
                                    padding='same',
                                    kernel_initializer=initializer,
                                    use_bias=False))

  result.add(tf.keras.layers.BatchNormalization())

  if apply_dropout:
      result.add(tf.keras.layers.Dropout(0.5))

  result.add(tf.keras.layers.ReLU())

  return result


up_model = upsample(3, 4)
up_result = up_model(down_result)
print (up_result.shape)
```

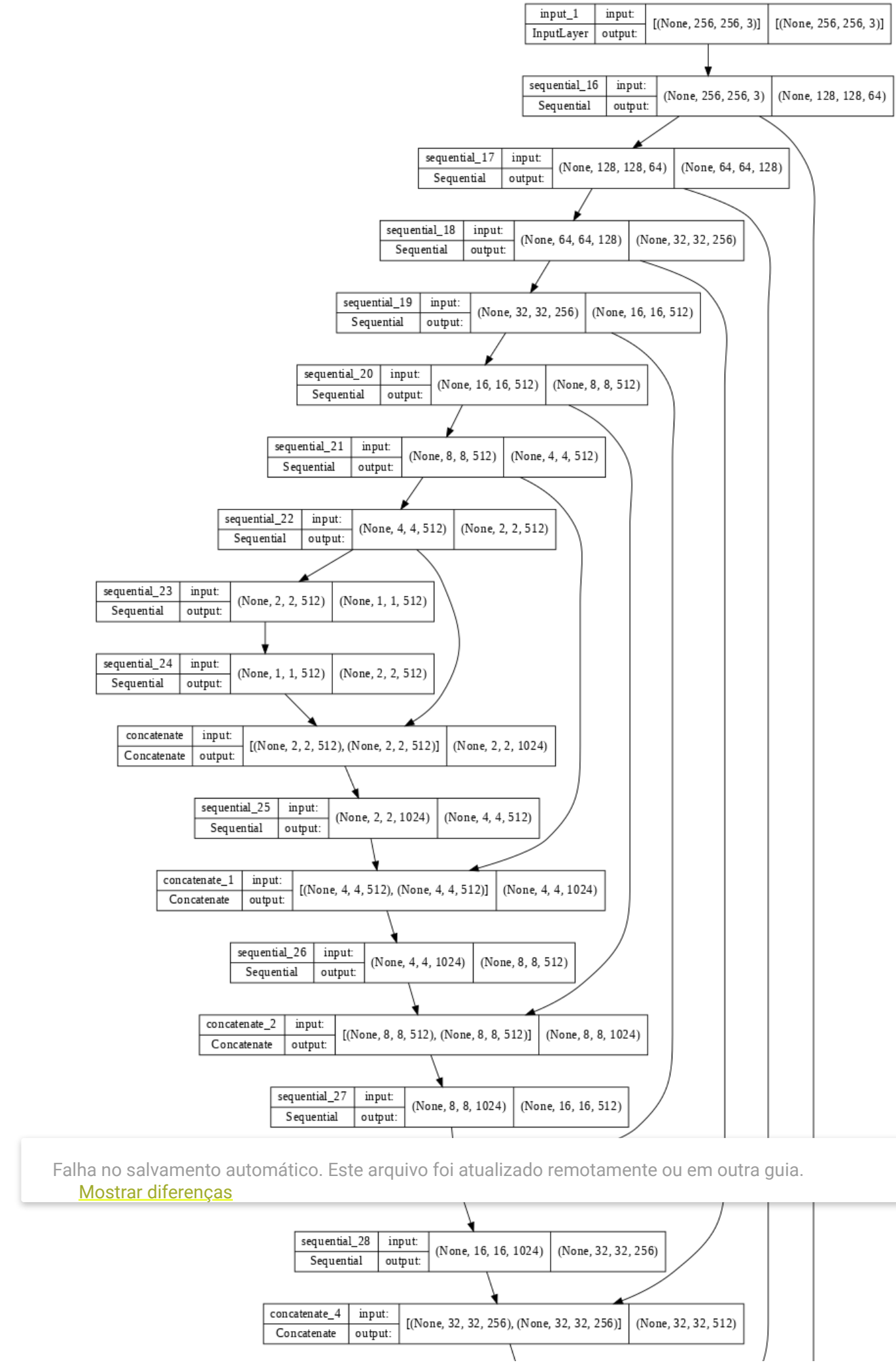
```
(1, 256, 256, 3)
```



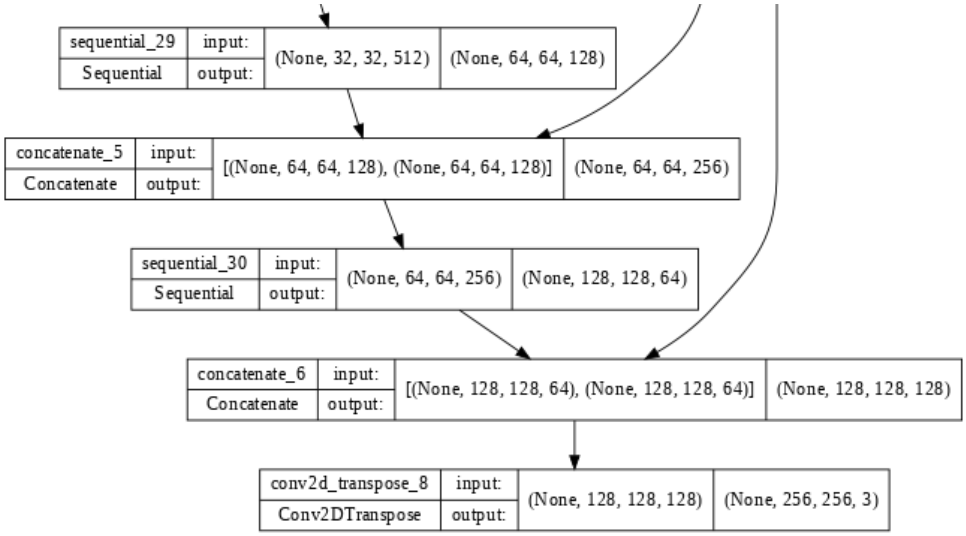
```
  down_stack = [
    downsample(64, 4, apply_batchnorm=False),  # (batch_size, 128, 128, 64)
    downsample(128, 4),  # (batch_size, 64, 64, 128)
    downsample(256, 4),  # (batch_size, 32, 32, 256)
    downsample(512, 4),  # (batch_size, 16, 16, 512)
```

```
    downsample(512, 4),  # (batch_size, 8, 8, 512)
    downsample(512, 4),  # (batch_size, 4, 4, 512)
    downsample(512, 4),  # (batch_size, 2, 2, 512)
    downsample(512, 4),  # (batch_size, 1, 1, 512)
  ]

  up_stack = [
    upsample(512, 4, apply_dropout=True),  # (batch_size, 2, 2, 1024)
    upsample(512, 4, apply_dropout=True),  # (batch_size, 4, 4, 1024)
    upsample(512, 4, apply_dropout=True),  # (batch_size, 8, 8, 1024)
    upsample(512, 4),  # (batch_size, 16, 16, 1024)
    upsample(256, 4),  # (batch_size, 32, 32, 512)
    upsample(128, 4),  # (batch_size, 64, 64, 256)
    upsample(64, 4),  # (batch_size, 128, 128, 128)
  ]

  initializer = tf.random_normal_initializer(0., 0.02)
  last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS, 4,
                                         strides=2,
                                         padding='same',
                                         kernel_initializer=initializer,
                                         activation='tanh')  # (batch_size, 256, 256, 3)

  x = inputs

  # Downsampling through the model
  skips = []
  for down in down_stack:
    x = down(x)
    skips.append(x)

  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    x = tf.keras.layers.Concatenate()([x, skip])

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)


generator = Generator()
tf.keras.utils.plot_model(generator, show_shapes=True, dpi=64)
```

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

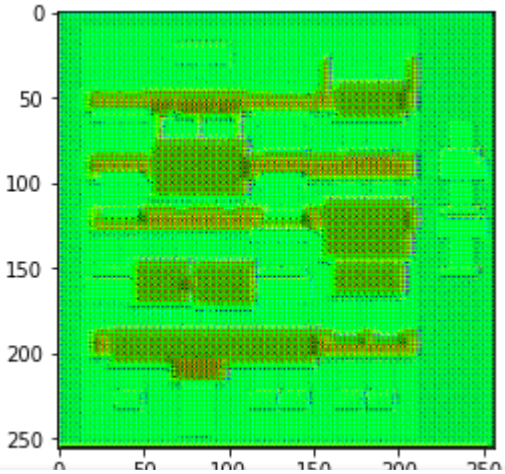


Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças




```
gen_output = generator(inp[tf.newaxis, ...], training=False)
plt.imshow(gen_output[0, ...])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats o
<matplotlib.image.AxesImage at 0x7f85fd21fed0>
```

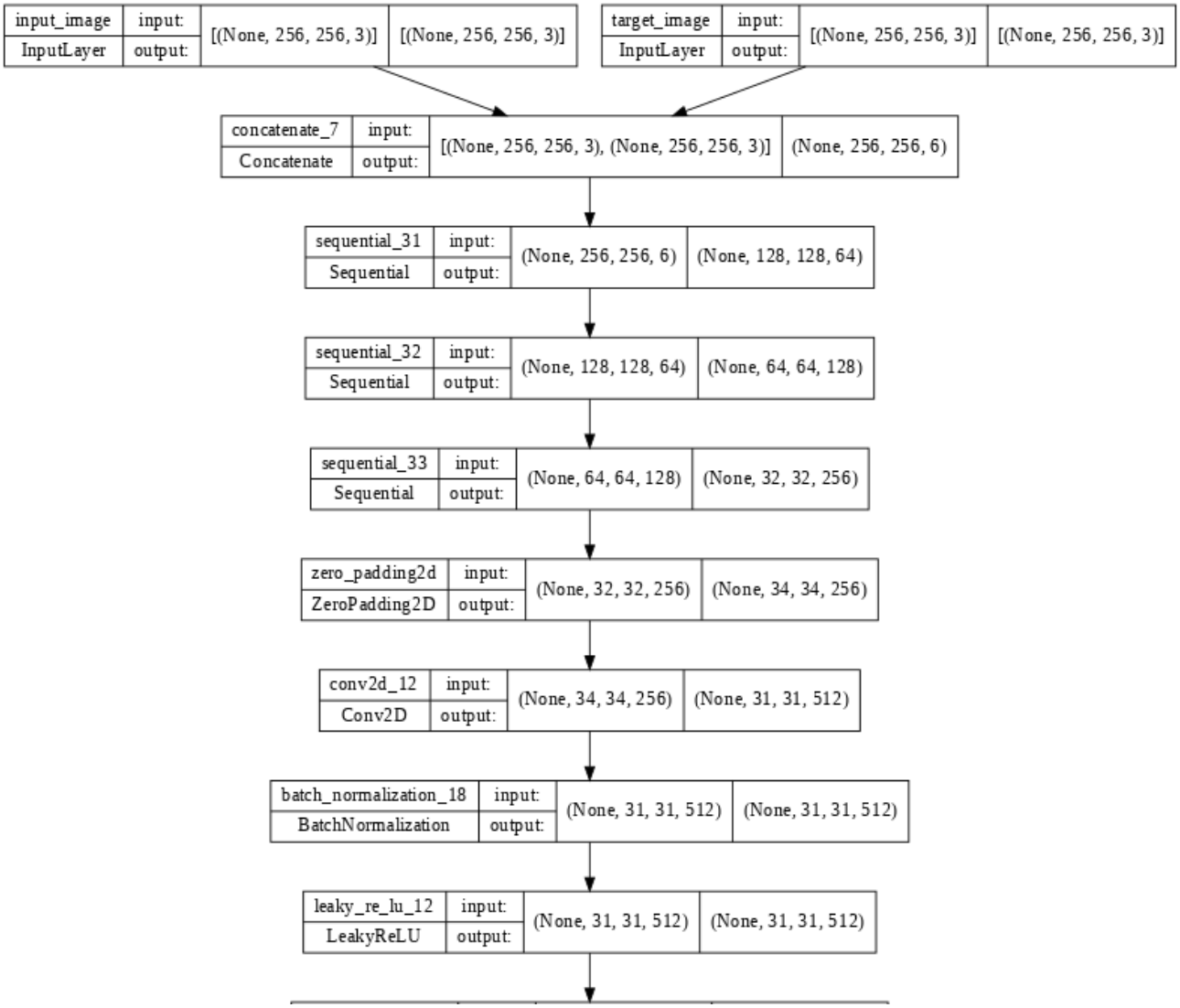



```
LAMBDA = 100
```

```
loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

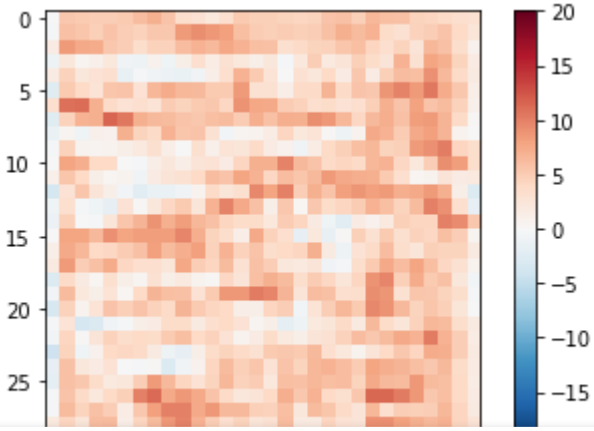
```
def generator_loss(disc_generated_output, gen_output, target):
```

```
    gan_loss = loss_object(tf.ones_like(disc_generated_output), disc_generated_output)

    # Mean absolute error
    l1_loss = tf.reduce_mean(tf.abs(target - gen_output))

    total_gen_loss = gan_loss + (LAMBDA * l1_loss)

    return total_gen_loss, gan_loss, l1_loss


def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)

    inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
    tar = tf.keras.layers.Input(shape=[256, 256, 3], name='target_image')

    x = tf.keras.layers.concatenate([inp, tar])  # (batch_size, 256, 256, channels*2)

    down1 = downsample(64, 4, False)(x)  # (batch_size, 128, 128, 64)
    down2 = downsample(128, 4)(down1)  # (batch_size, 64, 64, 128)
    down3 = downsample(256, 4)(down2)  # (batch_size, 32, 32, 256)

    zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)  # (batch_size, 34, 34, 256)
    conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                  kernel_initializer=initializer,
                                  use_bias=False)(zero_pad1)  # (batch_size, 31, 31, 512)

    batchnorm1 = tf.keras.layers.BatchNormalization()(conv)

    leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

    zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)  # (batch_size, 33, 33, 512)

    last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                  kernel_initializer=initializer)(zero_pad2)  # (batch_size,

    return tf.keras.Model(inputs=[inp, tar], outputs=last)


discriminator = Discriminator()
tf.keras.utils.plot_model(discriminator, show_shapes=True, dpi=64)
```




```
disc_out = discriminator([inp[tf.newaxis, ...], gen_output], training=False)
plt.imshow(disc_out[0, ..., -1], vmin=-20, vmax=20, cmap='RdBu_r')
plt.colorbar()
```

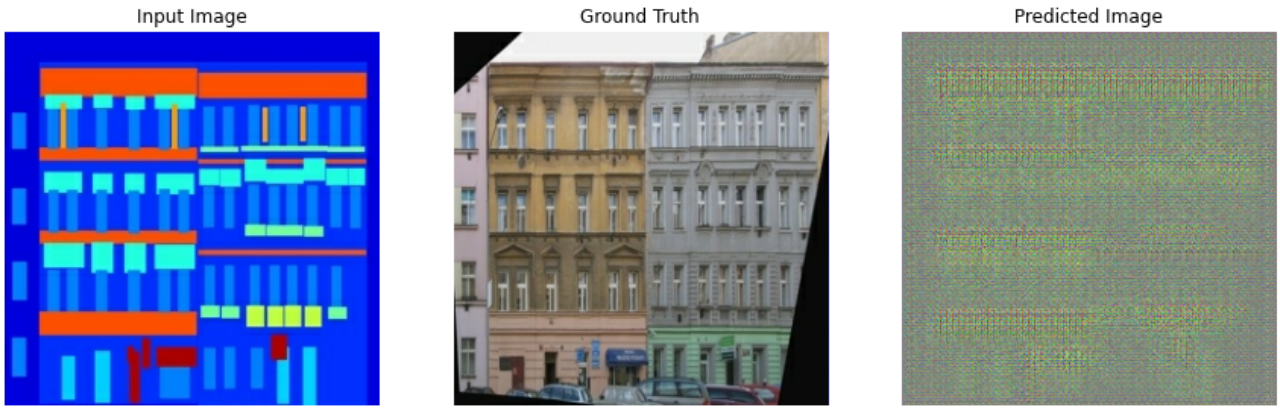
```
<matplotlib.colorbar.Colorbar at 0x7f86205b0f10>
```



Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

```
def discriminator_loss(disc_real_output, disc_generated_output):
  real_loss = loss_object(tf.ones_like(disc_real_output), disc_real_output)

  generated_loss = loss_object(tf.zeros_like(disc_generated_output), disc_generated_output
```

```
  total_disc_loss = real_loss + generated_loss

  return total_disc_loss

generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)


checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)


def generate_images(model, test_input, tar):
  prediction = model(test_input, training=True)
  plt.figure(figsize=(15, 15))

  display_list = [test_input[0], tar[0], prediction[0]]
  title = ['Input Image', 'Ground Truth', 'Predicted Image']

  for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.title(title[i])
    # Getting the pixel values in the [0, 1] range to plot.
    plt.imshow(display_list[i] * 0.5 + 0.5)
    plt.axis('off')
  plt.show()


for example_input, example_target in test_dataset.take(1):
  generate_images(generator, example_input, example_target)
```



Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças

```
log_dir="logs/"

summary_writer = tf.summary.create_file_writer(
  log_dir + "fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
```

```
@tf.function
def train_step(input_image, target, step):
  with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    gen_output = generator(input_image, training=True)

    disc_real_output = discriminator([input_image, target], training=True)
    disc_generated_output = discriminator([input_image, gen_output], training=True)

    gen_total_loss, gen_gan_loss, gen_l1_loss = generator_loss(disc_generated_output, gen_
    disc_loss = discriminator_loss(disc_real_output, disc_generated_output)

  generator_gradients = gen_tape.gradient(gen_total_loss,
                                          generator.trainable_variables)
  discriminator_gradients = disc_tape.gradient(disc_loss,
                                               discriminator.trainable_variables)

  generator_optimizer.apply_gradients(zip(generator_gradients,
                                          generator.trainable_variables))
  discriminator_optimizer.apply_gradients(zip(discriminator_gradients,
                                              discriminator.trainable_variables))

  with summary_writer.as_default():
    tf.summary.scalar('gen_total_loss', gen_total_loss, step=step//1000)
    tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=step//1000)
    tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=step//1000)
    tf.summary.scalar('disc_loss', disc_loss, step=step//1000)


def fit(train_ds, test_ds, steps):
  example_input, example_target = next(iter(test_ds.take(1)))
  start = time.time()

  for step, (input_image, target) in train_ds.repeat().take(steps).enumerate():
    if (step) % 1000 == 0:
      display.clear_output(wait=True)

      if step != 0:
        print(f'Time taken for 1000 steps: {time.time()-start:.2f} sec\n')

      start = time.time()

      generate_images(generator, example_input, example_target)
      print(f"Step: {step//1000}k")

    train_step(input_image, target, step)
```



```
    print('.', end='', flush=True)


    # Save (checkpoint) the model every 5k steps
    if (step + 1) % 5000 == 0:
      checkpoint.save(file_prefix=checkpoint_prefix)
```

```
%load_ext tensorboard
%tensorboard --logdir {log_dir}
```

No dashboards are active for the current data set.

Probable causes:

- You haven't written any data to your event files.
- TensorBoard can't find your event files.

If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the README and perhaps the TensorBoard tutorial.

If you think TensorBoard is configured properly, please see the section of the README devoted to missing data problems and consider filing an issue on GitHub.

Last reload: Jun 1, 2022, 1:14:36 PM

Log directory: logs/


```
fit(train_dataset, test_dataset, steps=40000)
```

```
Time taken for 1000 steps: 94.75 sec
```


```
Step: 39k
```

```
tensorboard dev upload --logdir {log_dir}
```

```
ERROR: Failed to launch TensorBoard (exited with 1).
Contents of stderr:
***** TensorBoard Uploader *****

This will upload your TensorBoard logs to https://tensorboard.dev/ from
the following directory:

logs/

This TensorBoard will be visible to everyone. Do not upload sensitive
data.

Your use of this service is subject to Google's Terms of Service
<https://policies.google.com/terms> and Privacy Policy
<https://policies.google.com/privacy>, and TensorBoard.dev's Terms of Service
<https://tensorboard.dev/policy/terms/>.

This notice will not be shown again while you are logged into the uploader.
To log out, run `tensorboard dev auth revoke`.

Traceback (most recent call last):
  File "/usr/local/bin/tensorboard", line 8, in <module>
    sys.exit(run_main())
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/main.py", line 46, in run
    app.run(tensorboard.main, flags_parser=tensorboard.configure)
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 312, in run
    _run_main(main, args)
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 258, in _run_main
    sys.exit(main(argv))
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/program.py", line 276, in
    return runner(self.flags) or 0
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/uploader/uploader_subcomm
```



```
m
  File "/usr/local/lib/python3.7/dist-packages/tensorboard/uploader/uploader_subcomm
    response = input("Continue? (yes/NO) ")
EOFError: EOF when reading a line
Contents of stdout:
Continue? (yes/NO)
```

```
display.IFrame(
    src="https://tensorboard.dev/experiment/lZ0C6FONROaUMfjYkVyJqw",
    width="100%",
    height="1000px")
```

```
ls {checkpoint_dir}
```

```
checkpoint                   ckpt-5.data-00000-of-00001
ckpt-1.data-00000-of-00001   ckpt-5.index
ckpt-1.index                 ckpt-6.data-00000-of-00001
ckpt-2.data-00000-of-00001   ckpt-6.index
ckpt-2.index                 ckpt-7.data-00000-of-00001
ckpt-3.data-00000-of-00001   ckpt-7.index
ckpt-3.index                 ckpt-8.data-00000-of-00001
ckpt-4.data-00000-of-00001   ckpt-8.index
ckpt-4.index
```

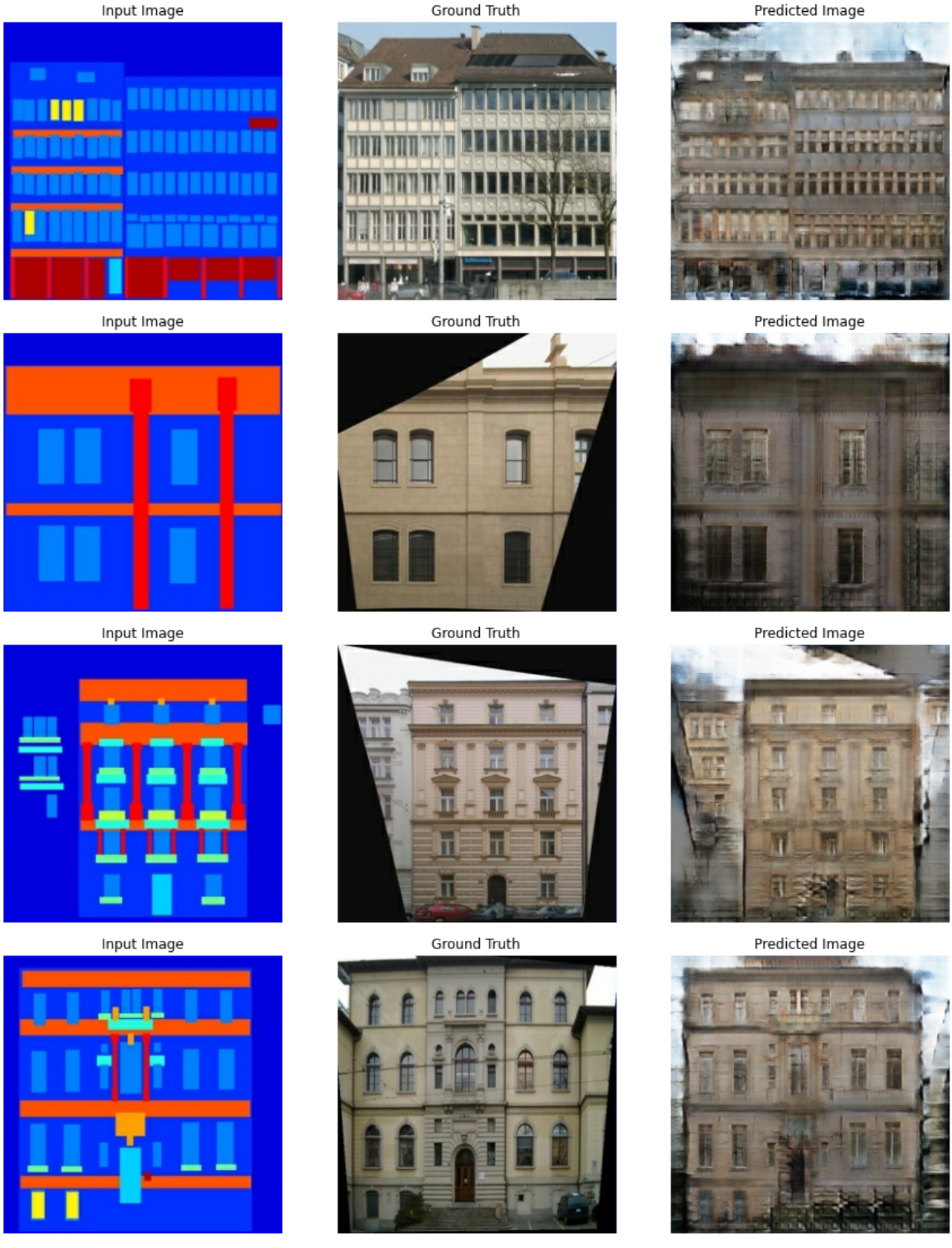


```
# Restoring the latest checkpoint in checkpoint_dir
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f85762f0e50>
```



```
# Run the trained model on a few examples from the test set
for inp, tar in test_dataset.take(5):
  generate_images(generator, inp, tar)
```




```
for inp, tar in test_dataset.take(10):
```

Falha no salvamento automático. Este arquivo foi atualizado remotamente ou em outra guia.
Mostrar diferenças