Lab 6 - PCC177/BCC406

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Modelos Generativos

Prof. Eduardo e Prof. Pedro

Aluna: Daniela Costa Terra

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"

Salvo com sucesso

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, rec
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model
```

Célula

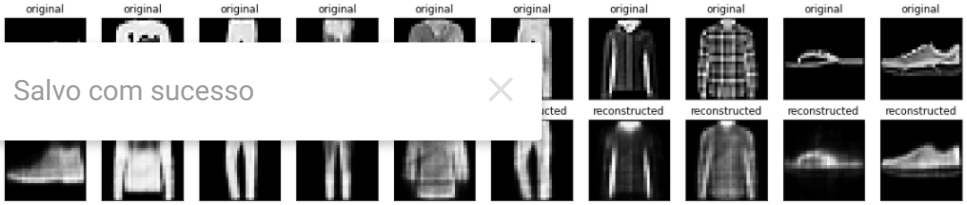
```
def load_image_train(
  input_image, real_i
  input_image, real_i
  input_image, real_i

  return input_image,

def load_image_test(i
  input_image, real_i
  input_image, real_i

  input_image, real_i

  return input_image,
```

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

```
(x_train, y_train), (x_test, y_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)
```

```
Downloading data from https://storage.googleapis.com/tenso
32768/29515 [================================] - 0s 0us/s
40960/29515 [========================================] -
Downloading data from https://storage.googleapis.com/tenso
26427392/26421880 [==============================] - 0s 0u
26435584/26421880 [==============================] - 0s 0u
Downloading data from https://storage.googleapis.com/tenso
16384/5148 [================================================
Downloading data from https://storage.googleapis.com/tenso
4423680/4422102 [==============================] - 0s 0us/
4431872/4422102 [==============================] - 0s 0us/
(60000, 28, 28)
(10000, 28, 28)
```

Exemplo de classes



Salvo com sucesso

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)
    return decoded

autoencoder = Autoencoder(latent_dim)

autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredErr


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

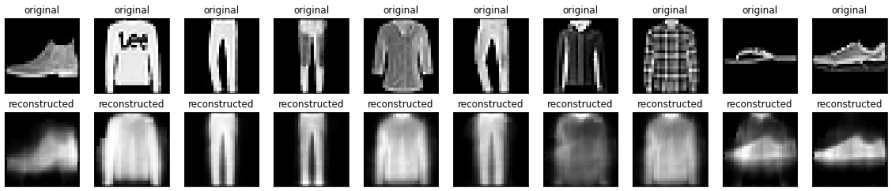
```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step -
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step -
1875/1875 [==============================] - 4s 2ms/step -
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step -
<keras.callbacks.History at 0x7fcc1b8c1110>
```

Salvo com sucesso

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

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


## Código para gráficos alterado (Daniela)
def showTest(decoded_imgs):
  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()

## Teste com latent_dim = 4
showTest(decoded_imgs)
```



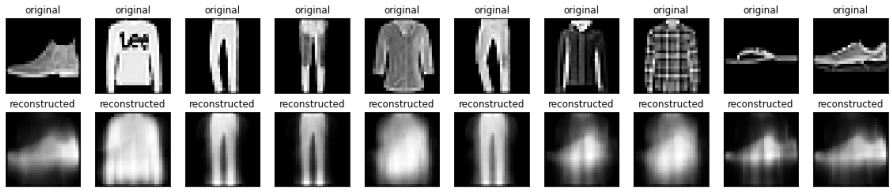
Salvo com sucesso

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

```
def encoderDecoder(latent_dim):
      autoencoder = Autoencoder(latent_dim)
      autoencoder.compile(optimizer='adam', loss=losses.MeanSqua
      autoencoder.fit(x_train, x_train,
                  epochs=10,
                  shuffle=True,
                  validation_data=(x_test, x_test))

      encoded_imgs = autoencoder.encoder(x_test).numpy()
      decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
      return (encoded_imgs, decoded_imgs)


## Vetor latente (latent_dim = 2):
```

```
(encoded_dim2, decoded_dim2) = encoderDecoder(2)
showTest(decoded_dim2)
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step -
```



Vetor latente (latent_dim = 8):

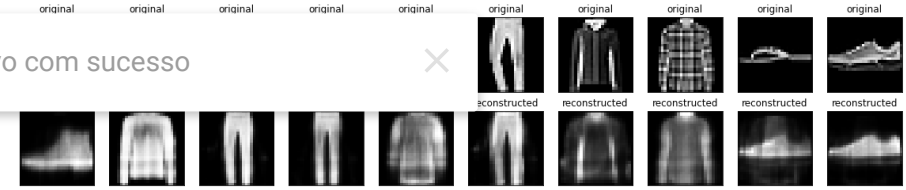
Salvo com sucesso

```
derDecoder(8)
showTest(decoded_dim8)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step -
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 5/10
```

```
## Vetor latente (latent_dim = 16):
(encoded_dim16, decoded_dim16) = encoderDecoder(16)
showTest(decoded_dim16)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step -
Epoch 3/10
1875/1875 [==============================] - 5s 2ms/step -
Epoch 4/10
1875/1875 [==============================] - 5s 2ms/step -
Epoch 5/10
1875/1875 [==============================] - 5s 2ms/step -
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step -
```



Salvo com sucesso

```
## Vetor latente (latent_dim = 64):
(encoded_dim64, decoded_dim64) = encoderDecoder(64)
showTest(decoded_dim64)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step -
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step -
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step -
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step -
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step -
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step -
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step -
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step -
Epoch 10/10
```

ToDo : Responda (15pt)

Escreva suas conclusões sobre os testes executados:

É perceptível para enconders com poucas dimensões latentes, como 2, 4, 8 e 16 que há uma tendência da rede em codificar de forma semelhante imagens semelhantes para objetos de mesma classe. É a idéia de continuidade: pontos que são próximos no espaço latente terão conteúdo similar após decodificação. Por exemplo, com 2, 4 e 8 dimensões a rede exibiu imagens de calças e [illegible]rma/textura. Para as duas últimas figuras, chinelo e tennis, as redes de até 16 dimensões também exibem as imagens correspondentes com a mesma aparência (de tennis). Apenas com 64 dimensões no enconder a imagem desses objetos são diferenciados e a rede começa produzir imagens mais próximas da imagem real de entrada (completude).

Salvo com sucesso

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

Base de dados detalhada no site: timeseriesclassification.com.

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

	0	1	2	3	4	
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234
2			30	-4.584095	-4.187449	-3.151
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802

5 rows × 141 columns

Salvo com sucesso

```
# 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_sp
    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)
```

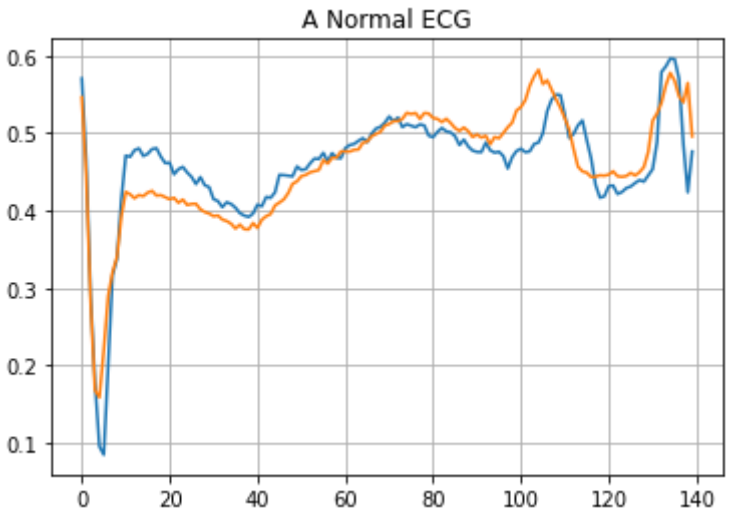
Vamos separar os batimentos normais (label 1) para treinar o Autoencoder.

```
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]

anomalous_train_data = train_data[~train_labels]
anomalous_test_data = test_data[~test_labels]
```

Plote um batimento normal.

```
plt.grid()
                                                in_data[0], normal_train_data
plt.show()
```

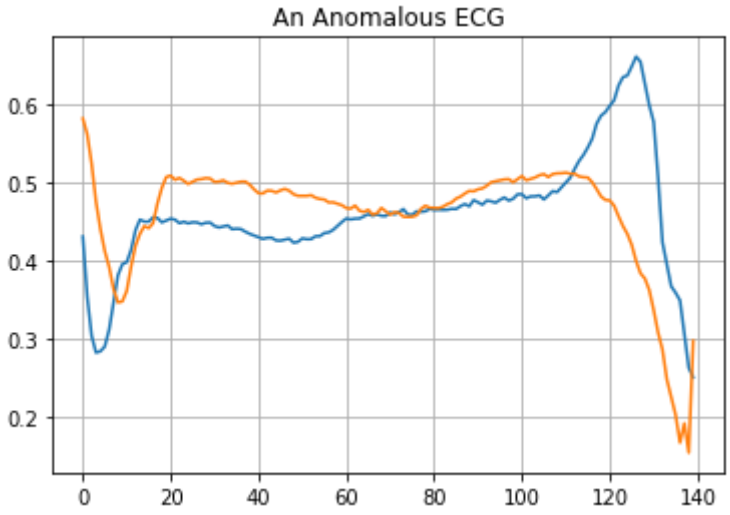
Salvo com sucesso



Plote um batimento anômalo.

```
plt.grid()
```

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

```
## Expande dim dos dados de entrada: para cada batimento são 140
                                    normal_train_data, axis=2)
                                    ormal_test_data, axis=2)

normal_train_data.shape, normal_test_data.shape
```

```
(TensorShape([2359, 140, 1]), TensorShape([560, 140, 1]))
```

Salvo com sucesso

Testes Autoencoders: camadas densas (D) e convoluções (C) testadas em paralelo

```
#  Anomaly detector (Dense net)
class AnomalyDenseDetector(Model):
  def __init__(self, units_dense1, units_dense2, latent_dim):
    super(AnomalyDenseDetector, self).__init__()
    self.units_dense1 = units_dense1
    self.units_dense2 = units_dense2
    self.latent_dim = latent_dim
    # Todo: crie o encoder, com um gargalo
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
```

```
      layers.Dense(units_dense1, activation='relu'),
      layers.Dense(units_dense2, activation='relu'),
      layers.Dense(latent_dim, activation='relu'),
    ])
    # Todo: crie as camadas do decoder
    self.decoder = tf.keras.Sequential([
      layers.Dense(units_dense2, activation='relu'),
      layers.Dense(units_dense1, activation='relu'),
      layers.Dense(140, activation='sigmoid'),
      layers.Reshape((140, 1))
    ])

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

#  Anomaly detector (Convolucional net)
class AnomalyConvDetector(Model):
  def __init__(self, n_filters1, n_filters2):   # dimensões late
    super(AnomalyConvDetector, self).__init__()
    self.n_filters1 = n_filters1
    self.n_filters2 = n_filters2
    # Todo: crie o encoder, com um gargalo
    self.encoder = tf.keras.Sequential([
      layers.Conv1D(n_filters1, 2, strides=2),
      layers.Conv1D(n_filters2, 2, strides=2,activation='relu'),
      layers.Conv1D(1,1,activation='relu'),
    ])
    # Todo: crie as camadas do decoder
    self.decoder = tf.keras.Sequential([
      layers.Conv1DTranspose(n_filters2, 2, strides=2, activatio
      layers.Conv1DTranspose(n_filters1, 2, strides=2),

      layers.Dense(140, activation='sigmoid'),
      layers.Reshape((140, 1))
    ])

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


autoencoderD = AnomalyDenseDetector(280, 110, 30)

autoencoderC = AnomalyConvDetector(256, 512)


autoencoderD.compile(optimizer='adam', loss='mae')

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

Salvo com sucesso

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

```
historyD = autoencoderD.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 111ms/step - los
Epoch 2/20
5/5 [==============================] - 0s 36ms/step - loss
Epoch 3/20
5/5 [==============================] - 0s 26ms/step - loss
Epoch 4/20
5/5 [==============================] - 0s 26ms/step - loss
Epoch 5/20
5/5 [==============================] - 0s 27ms/step - loss
Epoch 6/20
5/5 [==============================] - 0s 25ms/step - loss
Epoch 7/20
5/5 [==============================] - 0s 27ms/step - loss
Epoch 8/20
5/5 [==============================] - 0s 26ms/step - loss
Epoch 9/20
5/5 [==============================] - 0s 26ms/step - loss
Epoch 10/20
5/5 [==============================] - 0s 25ms/step - loss
Epoch 11/20
5/5 [==============================] - 0s 27ms/step - loss
```

Salvo com sucesso

```
5/5 [==============================] - 0s 31ms/step - loss
Epoch 14/20
5/5 [==============================] - 0s 28ms/step - loss
Epoch 15/20
5/5 [==============================] - 0s 25ms/step - loss
Epoch 16/20
5/5 [==============================] - 0s 29ms/step - loss
Epoch 17/20
5/5 [==============================] - 0s 26ms/step - loss
Epoch 18/20
5/5 [==============================] - 0s 26ms/step - loss
Epoch 19/20
5/5 [==============================] - 0s 28ms/step - loss
Epoch 20/20
5/5 [==============================] - 0s 27ms/step - loss
```

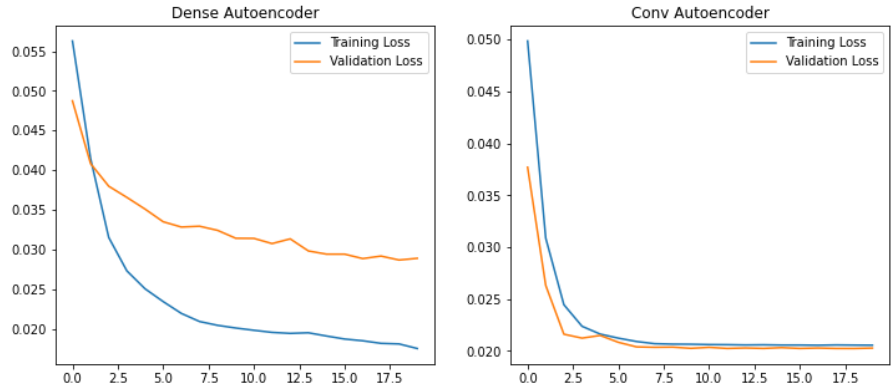
```
historyC = autoencoderC.fit(normal_train_data, normal_train_data
          epochs=20,
          batch_size=512,
```

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

```
Epoch 1/20
5/5 [==============================] - 18s 3s/step - loss:
Epoch 2/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 3/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 4/20
5/5 [==============================] - 16s 3s/step - loss:
Epoch 5/20
5/5 [==============================] - 15s 3s/step - loss:
Epoch 6/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 7/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 8/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 9/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 10/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 11/20
5/5 [==============================] - 15s 3s/step - loss:
Epoch 12/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 13/20
5/5 [==============================] - 15s 3s/step - loss:
Epoch 14/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 15/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 16/20
                                    ====] - 15s 3s/step - loss:
Salvo com sucesso
                                    ====] - 14s 3s/step - loss:
Epoch 18/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 19/20
5/5 [==============================] - 14s 3s/step - loss:
Epoch 20/20
5/5 [==============================] - 14s 3s/step - loss:
```

```
_, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,5))

ax1.plot(historyD.history["loss"], label="Training Loss ")
ax1.plot(historyD.history["val_loss"], label="Validation Loss")
ax1.legend()
ax1.set_title("Dense Autoencoder")

ax2.plot(historyC.history["loss"], label="Training Loss ")
ax2.plot(historyC.history["val_loss"], label="Validation Loss")
ax2.legend()
ax2.set_title("Conv Autoencoder")
```

```
Text(0.5, 1.0, 'Conv Autoencoder')
```

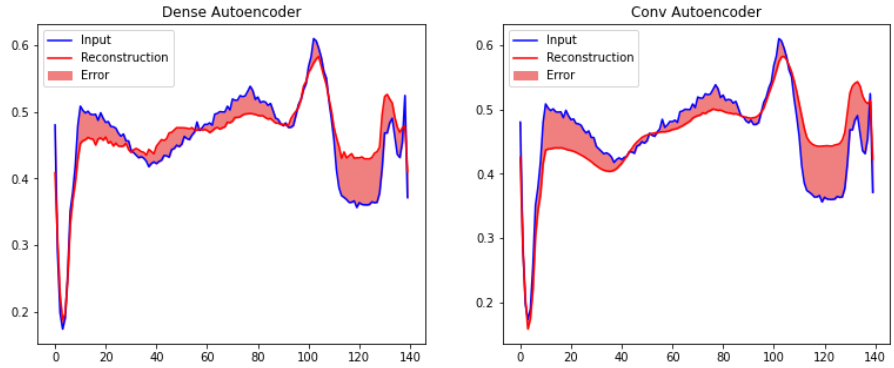


Você vai considerar um batimento como anômalo se ele divergir masi que um desvio padrão das amostras normais. Primeiro, vamos plotar um batimento normal a partir da base de treino e sua reconstrução. Assim, poderemos calcular o erro de re-construção.

```
## Dense Autoencoder
encoded_dataD = autoencoderD.encoder(normal_test_data).numpy()
decoded_dataD = autoencoderD.decoder(encoded_dataD).numpy()

                                    e(normal_test_data[0])
                                    ecoded_dataD[0])

_, (ax1, ax2) = plt.subplots(1, 2, figsize=(13,5))
ax1.plot(normal_test_data_sample, 'b')
ax1.plot(decoded_data_sampleD, 'r')
ax1.fill_between(np.arange(140), decoded_data_sampleD, normal_te
ax1.legend(labels=["Input", "Reconstruction", "Error"])
ax1.set_title("Dense Autoencoder")

##  Conv Autoencoder
encoded_dataC = autoencoderC.encoder(normal_test_data).numpy()
decoded_dataC = autoencoderC.decoder(encoded_dataC).numpy()
decoded_data_sampleC = tf.squeeze(decoded_dataC[0])

ax2.plot(normal_test_data_sample, 'b')
ax2.plot(decoded_data_sampleC, 'r')
ax2.fill_between(np.arange(140), decoded_data_sampleC, normal_te
ax2.legend(labels=["Input", "Reconstruction", "Error"])
ax2.set_title("Conv Autoencoder")
```

Salvo com sucesso

```
Text(0.5, 1.0, 'Conv Autoencoder')
```

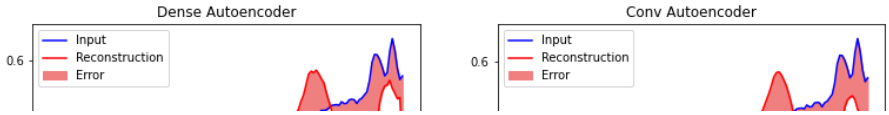


Vamos fazer o mesmo para um batimento anômalo.

```
encoded_dataD = autoencoderD.encoder(anomalous_test_data).numpy(
decoded_dataD = autoencoderD.decoder(encoded_dataD).numpy()

_, (ax1, ax2) = plt.subplots(1, 2, figsize=(13,5))

ax1.plot(tf.squeeze(anomalous_test_data[0]), 'b')
ax1.plot(tf.squeeze(decoded_dataD[0]), 'r')
ax1.fill_between(np.arange(140), tf.squeeze(decoded_dataD[0]), t
ax1.legend(labels=["Input", "Reconstruction", "Error"])
ax1.set_title("Dense Autoencoder")

##  Conv Autoencoder
                                                (anomalous_test_data).numpy(
                                                r(encoded_dataC).numpy()

ax2.plot(tf.squeeze(anomalous_test_data[0]), 'b')
ax2.plot(tf.squeeze(decoded_dataC[0]), 'r')
ax2.fill_between(np.arange(140), tf.squeeze(decoded_dataC[0]), t
ax2.legend(labels=["Input", "Reconstruction", "Error"])
ax2.set_title("Conv Autoencoder")
```

Salvo com sucesso

```
Text(0.5, 1.0, 'Conv Autoencoder')
```



Detectando as anomalias

Vamos detectar as anomalis se o erro de reconstrução for maior que um limiar. Aqui, vamos 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.

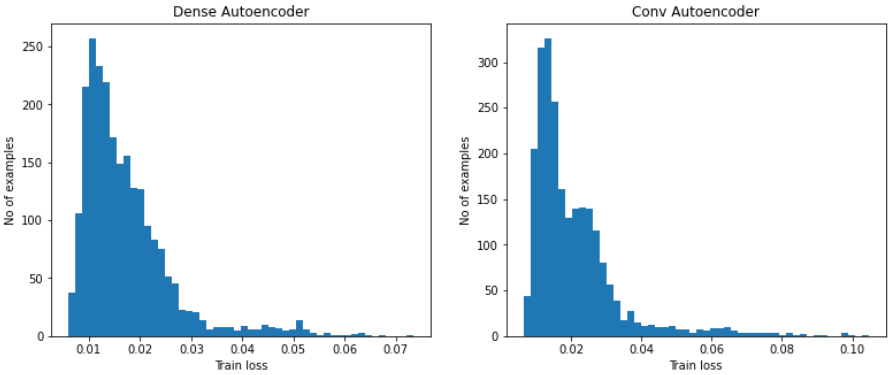
Plota erro de reconstrução de batimentos normais do treino

```
reconstructionsD = autoencoderD.predict(normal_train_data)
train_lossD = tf.keras.losses.mae(tf.squeeze(reconstructionsD),

reconstructionsC = autoencoderC.predict(normal_train_data)
train_lossC = tf.keras.losses.mae(tf.squeeze(reconstructionsC),

_, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 5))
ax1.hist(train_lossD[None,:], bins=50)
ax1.set_xlabel("Train loss")
ax1.set_ylabel("No of examples")
ax1.set_title("Dense Autoencoder")

ax2.hist(train_lossC[None,:], bins=50)
```

Salvo com sucesso

```
ax2.set_title("Conv Autoencoder")
```

```
Text(0.5, 1.0, 'Conv Autoencoder')
```



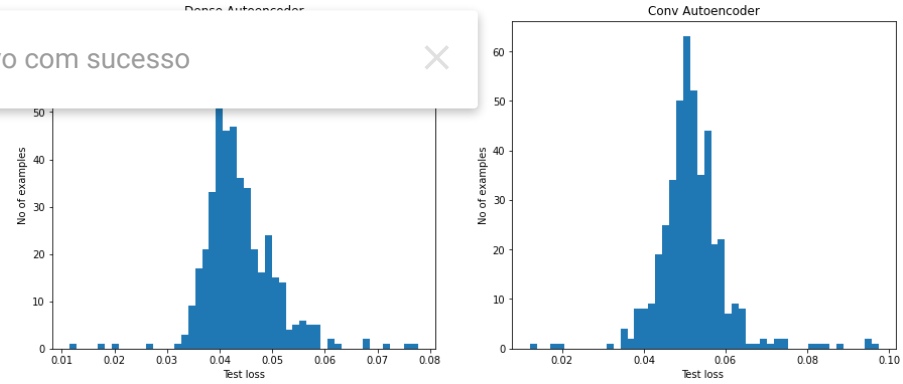
Escolha do limiar.

```
thresholdD = np.mean(train_lossD) + np.std(train_lossD)
print("Threshold Dense: ", thresholdD)

thresholdC = np.mean(train_lossC) + np.std(train_lossC)
print("Threshold Conv: ", thresholdC)
```

```
Threshold Dense:  0.026340492
Threshold Conv:  0.03315819
```

```
reconstructionsD = autoencoderD.predict(anomalous_test_data)
test_lossD = tf.keras.losses.mae(tf.squeeze(reconstructionsD), t

_, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6))
ax1.hist(test_lossD[None, :], bins=50)
ax1.set_xlabel("Test loss")
ax1.set_ylabel("No of examples")
ax1.set_title("Dense Autoencoder")

reconstructionsC = autoencoderC.predict(anomalous_test_data)
test_lossC = tf.keras.losses.mae(tf.squeeze(reconstructionsC), t

ax2.hist(test_lossC[None, :], bins=50)
ax2.set_xlabel("Test loss ")
ax2.set_ylabel("No of examples")
ax2.set_title("Conv Autoencoder")
```

```
Text(0.5, 1.0, 'Conv Autoencoder')
```

Classificação.

```
def predict(model, data, threshold):
  reconstructions = model(data)
```

```
  loss = tf.keras.losses.mae(tf.squeeze(reconstructions), tf.squ
  return tf.math.less(loss, threshold)

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

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

```
predsD = predict(autoencoderD, test_data, thresholdD)
print_stats(predsD, test_labels, "Dense")

predsC = predict(autoencoderC, test_data, thresholdC)
print_stats(predsC, test_labels, "Conv")
```

```
Accuracy(Dense) = 0.935
Precision(Dense) = 0.9940119760479041
Recall(Dense) = 0.8892857142857142

Accuracy(Conv) = 0.944
Precision(Conv) = 0.9921875
Recall(Conv) = 0.9071428571428571
```

Parte III: Redes Generativas Adversariais (40pt)

Salvo com sucesso

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)

```
# ToDO : Criar 3 máscaras e gerar 3 saídas com a pix2pix para o
```

Cópia do código pix2pix

(código executado com GPU)

```
import tensorflow as tf

import os
import pathlib
import time
import datetime

from matplotlib import pyplot as plt
from IPython import display


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
dataset_name = "facades" #@param ["cityscapes", "e...dbags"
```

dataset_name: facades

```
_URL = f'http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/{da

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

Salvo com sucesso

```
_zip)

PATH = path_to_zip.parent/dataset_name
```

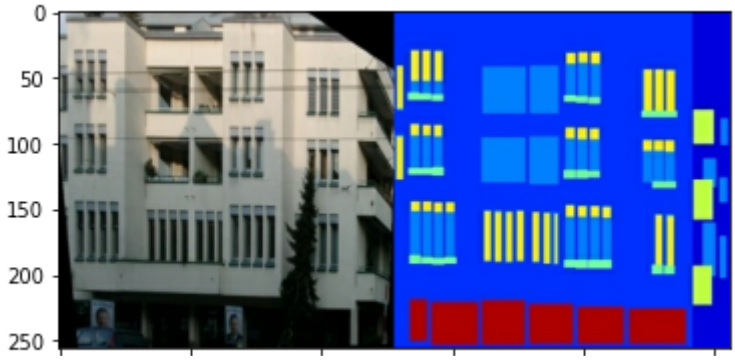
```
Downloading data from http://efrosgans.eecs.berkeley.edu/p
30171136/30168306 [==============================] - 18s 1
30179328/30168306 [==============================] - 18s 1
```

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

```
[PosixPath('/root/.keras/datasets/facades'),
 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)
plt.figure()
plt.imshow(sample_image)
```

```
(256, 512, 3)
<matplotlib.image.AxesImage at 0x7ffb546de250>
```

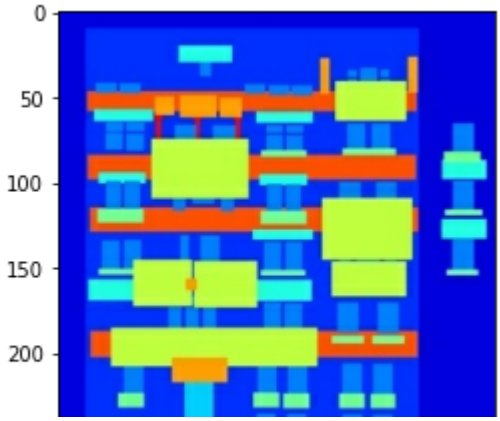



```
def load(image_file):
  # Read and decode an image file to a uint8 tensor
  image = tf.io.read_file(image_file)
  image = tf.io.decode_jpeg(image)

  # 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
```

Salvo com sucesso

```
0.jpg'))
display the images
plt.figure()
plt.imshow(inp / 255.0)
plt.figure()
plt.imshow(re / 255.0)
```

```
<matplotlib.image.AxesImage at 0x7ffb40222510>
```



Pré-processamento:


```
# The facade training set consist of 400 images
BUFFER_SIZE = 400
# The batch size of 1 produced better results for the U-Net in t
BATCH_SIZE = 1
# Each image is 256x256 in size
IMG_WIDTH = 256
IMG_HEIGHT = 256
```

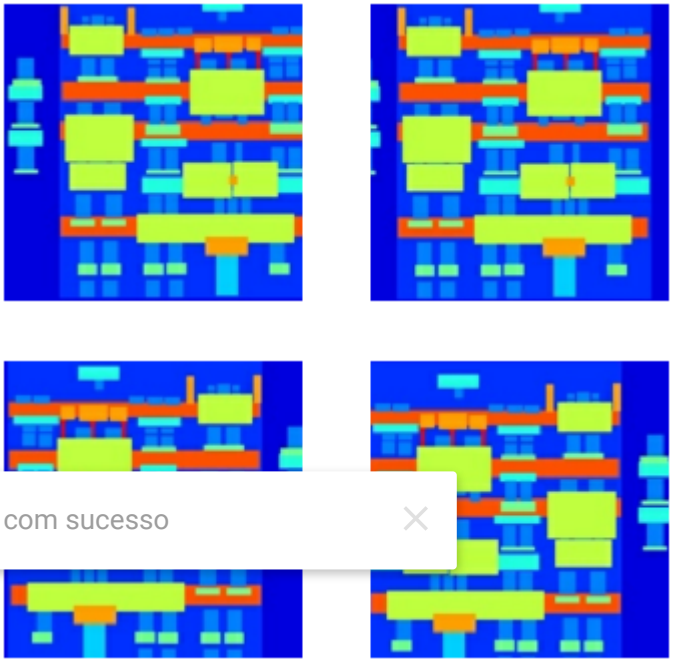

```
OUTPUT_CHANNELS = 3
```


```
def resize(input_image, real_image, height, width):
  input_image = tf.image.resize(input_image, [height, width],
                                method=tf.image.ResizeMethod.NEA
  real_image = tf.image.resize(real_image, [height, width],
                               od=tf.image.ResizeMethod.NEAR
```

Salvo com sucesso

```
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,
```

```
  # 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.figure(figsize=(6, 6))
for i in range(4):
  rj_inp, rj_re = random_jitter(inp, re)
  plt.subplot(2, 2, i + 1)
  plt.imshow(rj_inp / 255.0)
  plt.axis('off')
plt.show()
```



Salvo com sucesso

Carrega bancos:

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

```
train_dataset = tf.data.Dataset.list_files(str(PATH / 'train/*.j
train_dataset = train_dataset.map(load_image_train,
                                  num_parallel_calls=tf.data.AUT
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/*.j
except tf.errors.InvalidArgumentError:
  test_dataset = tf.data.Dataset.list_files(str(PATH / 'val/*.jp
test_dataset = test_dataset.map(load_image_test)
test_dataset = test_dataset.batch(BATCH_SIZE)
```

Gerador (downsample e upsample):

```
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='
                             kernel_initializer=initializer, use

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

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

Salvo com sucesso

```
#teste downsample
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=initializ
                                    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


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

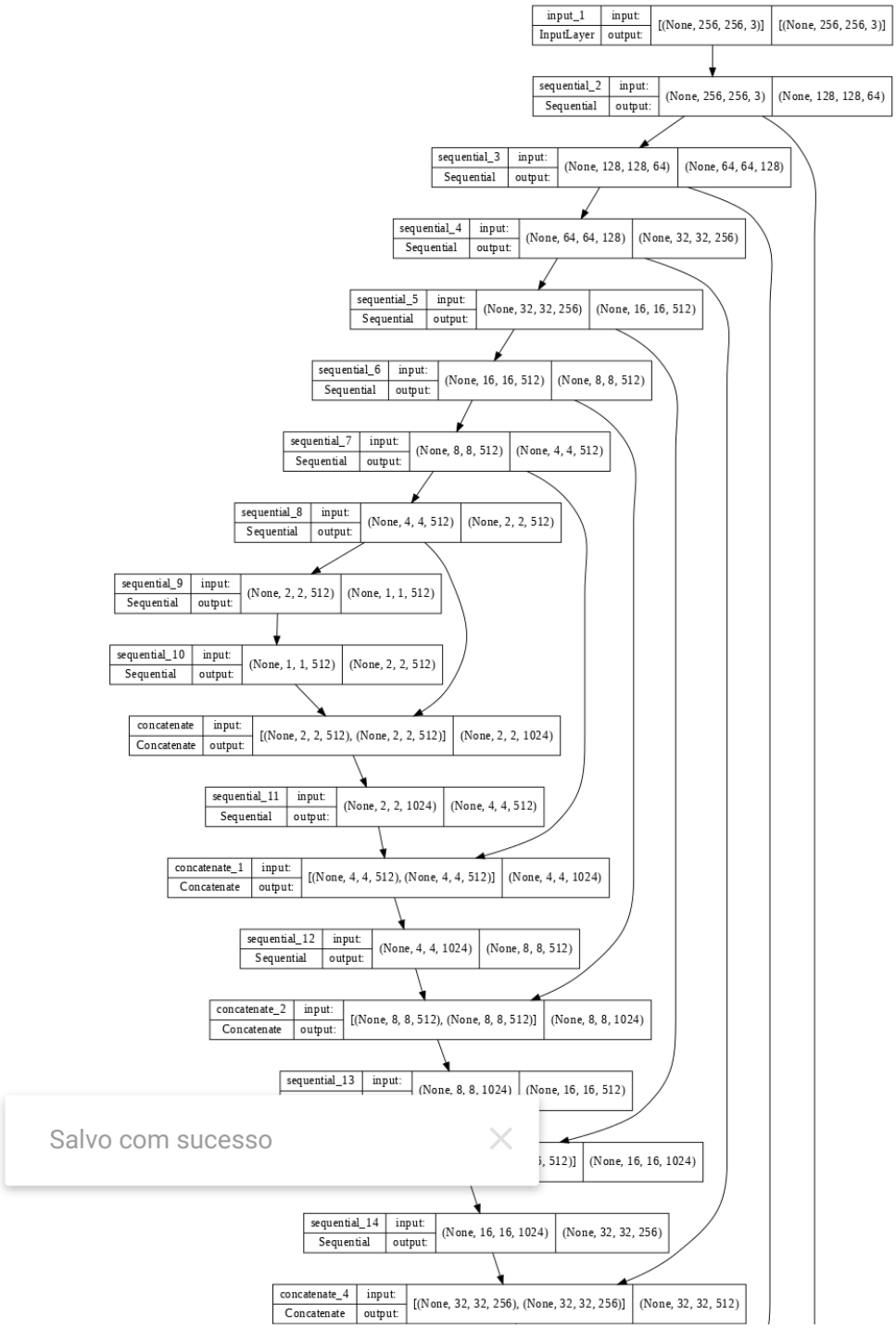
```
(1, 256, 256, 3)
```

```
# Define o Gerador:
def Generator():
  inputs = tf.keras.layers.Input(shape=[256, 256, 3])

  down_stack = [
    downsample(64, 4, apply_batchnorm=False),  # (batch_size, 12
    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,
    upsample(512, 4, apply_dropout=True),  # (batch_size, 4, 4,
    upsample(512, 4, apply_dropout=True),  # (batch_size, 8, 8,
    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=init
                                         activation='tanh')  # (

  x = inputs

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

Salvo com sucesso

```
    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)
```

Salvo com sucesso



Salvo com sucesso ×

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

```
Clipping input data to the valid range for imshow with RGB
<matplotlib.image.AxesImage at 0x7ffad622cb90>
```

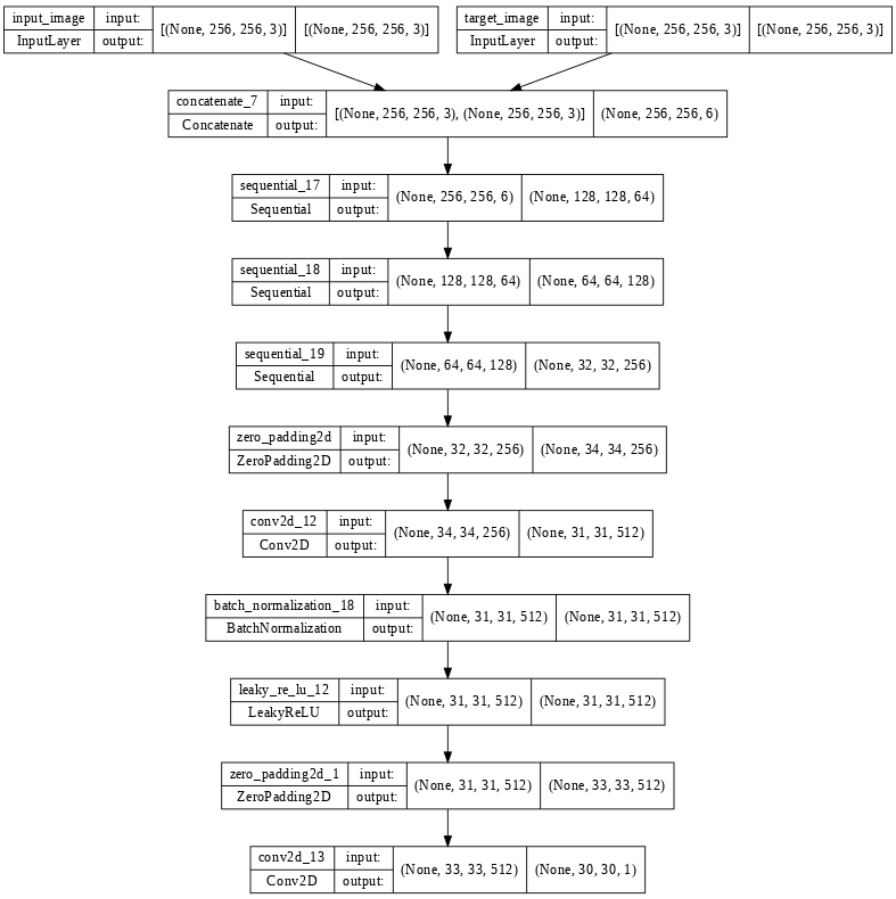

```
# Função de perda do gerador:
LAMBDA = 100
loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=Tru
```

```
def generator_loss(disc_generated_output, gen_output, target):
  gan_loss = loss_object(tf.ones_like(disc_generated_output), di

  # 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
```

Discriminador:

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

  inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_i
  tar = tf.keras.layers.Input(shape=[256, 256, 3], name='target_

  x = tf.keras.layers.concatenate([inp, tar])  # (batch_size, 25

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

  zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)  # (batch_s
  conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                kernel_initializer=initializer,
                                use_bias=False)(zero_pad1)  # (b

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

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

  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)  # (ba

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

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


discriminator = Discriminator()
tf.keras.utils.plot_model(discriminator,·show_shapes=True,·dpi=6
```

Salvo com sucesso



Salvo com sucesso

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

```
<matplotlib.colorbar.Colorbar at 0x7ffad61e5550>
```


```
# Função de perda (Discrimidador)

def discriminator_loss(disc_real_output, disc_generated_output):
  real_loss = loss_object(tf.ones_like(disc_real_output), disc_r
  generated_loss = loss_object(tf.zeros_like(disc_generated_outp
  total_disc_loss = real_loss + generated_loss
  return total_disc_loss

          0     5    10    15    20    25

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


#checkpoint_dir = './training_checkpoints'
checkpoint_dir ='/content/drive/MyDrive/disciplinasDoutorado/PCC
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_o
                                 discriminator_optimizer=discrim
                                 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)
```

Treinamento:

```
log_dir="/content/drive/MyDrive/disciplinasDoutorado/PCC177-2022

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



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

    disc_real_output = discriminator([input_image, target], trai
    disc_generated_output = discriminator([input_image, gen_outp

    gen_total_loss, gen_gan_loss, gen_l1_loss = generator_loss(d
    disc_loss = discriminator_loss(disc_real_output, disc_genera

  generator_gradients = gen_tape.gradient(gen_total_loss,
                                          generator.trainable_va
  discriminator_gradients = disc_tape.gradient(disc_loss,
                                               discriminator.tra

  generator_optimizer.apply_gradients(zip(generator_gradients,
                                          generator.trainable_va
  discriminator_optimizer.apply_gradients(zip(discriminator_grad
                                              discriminator.trai

  with summary_writer.as_default():
    tf.summary.scalar('gen_total_loss', gen_total_loss, step=ste
    tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=step//1
    tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=step//100
    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(step
    if (step) % 1000 == 0:
      display.clear_output(wait=True)

      if step != 0:
```

Salvo com sucesso

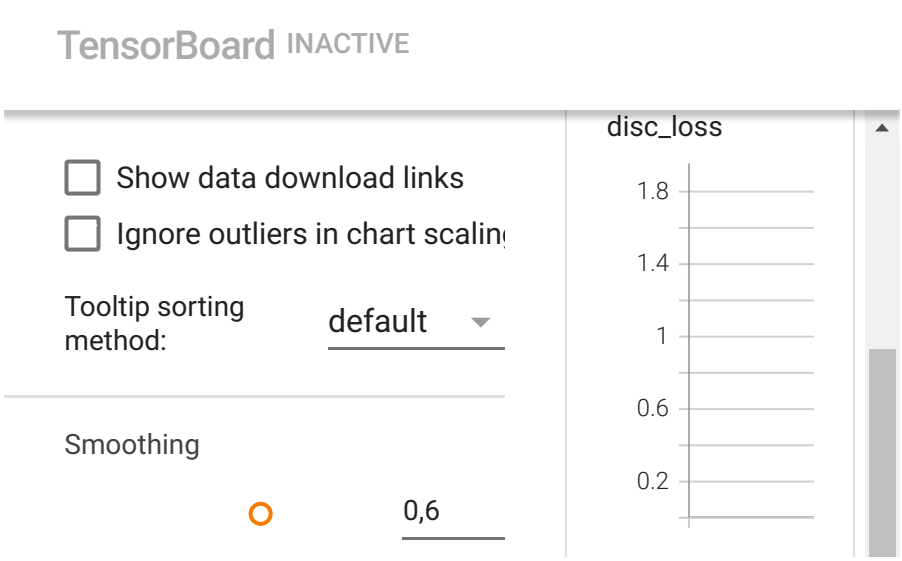
```
      print(f'Time taken for 1000 steps: {time.time()-start:.2

    start = time.time()

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

  train_step(input_image, target, step)

  # Training step
  if (step+1) % 10 == 0:
    print('.', end='', flush=True)


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

```
#!rm -rf ./logs/
```

```
%load_ext tensorboard
```

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

Salvo com sucesso




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

```
Time taken for 1000 steps: 95.24 sec
```


```
Step: 39k
.............................................................
```

Salvo com sucesso

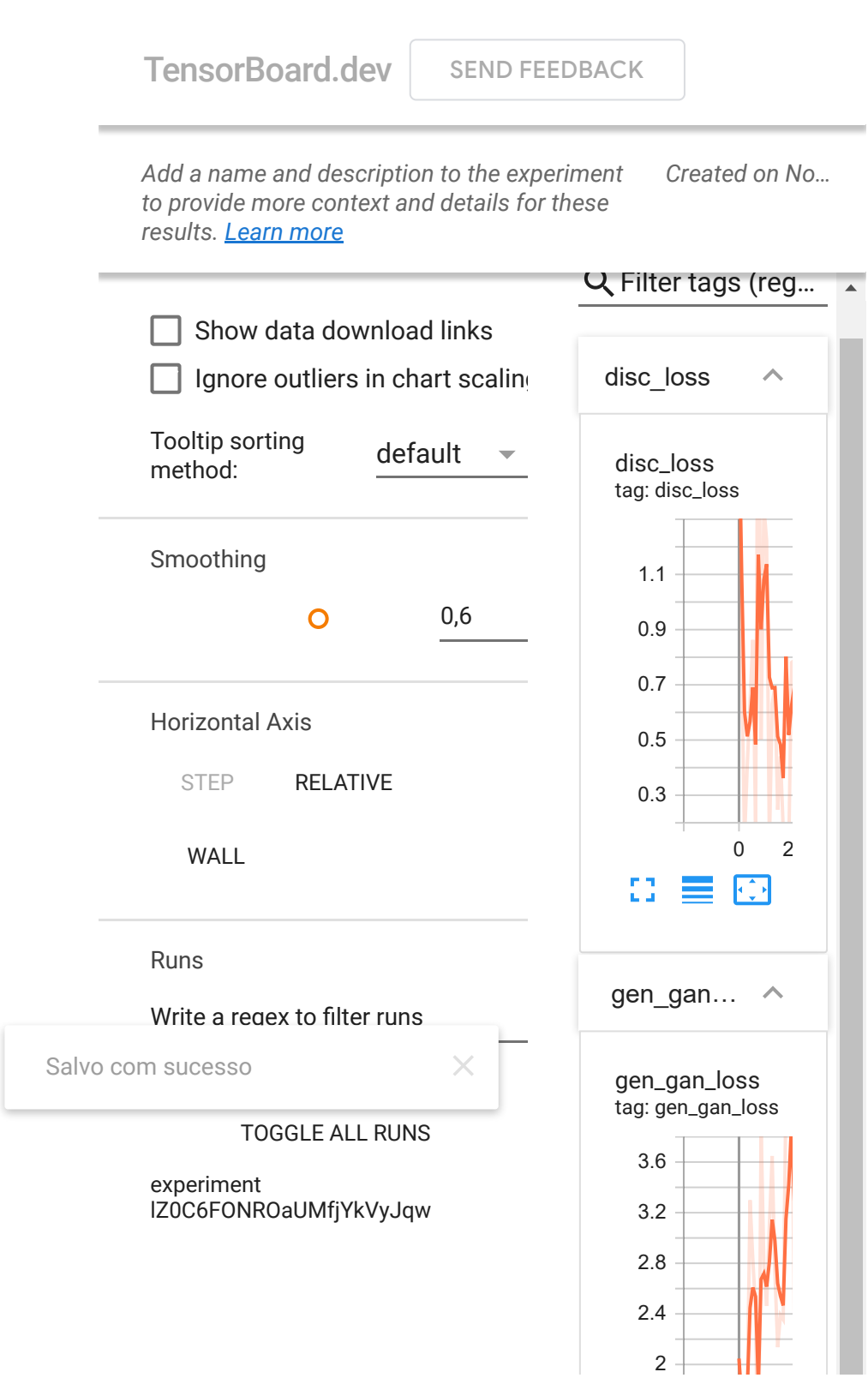

```
!tensorboard dev upload --logdir '{log_dir}'
```

```
***** TensorBoard Uploader *****

This will upload your TensorBoard logs to https://tensorbo
the following directory:

/content/drive/MyDrive/disciplinasDoutorado/PCC177-2022-1(

This TensorBoard will be visible to everyone. Do not uploa
data.

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

This notice will not be shown again while you are logged i
```

```
To log out, run `tensorboard dev auth revoke`.

Continue? (yes/NO) yes

Please visit this URL to authorize this application: https
Enter the authorization code: 4/1AdQt8qiJd_YK8G3mqEr-F7AsN

Upload started and will continue reading any new data as i

To stop uploading, press Ctrl-C.

New experiment created. View your TensorBoard at: https://

[2022-07-04T13:27:49] Started scanning logdir.
E0704 13:28:56.799176 140024240031616 uploader.py:564] Upl
        status = StatusCode.RESOURCE_EXHAUSTED
        details = "Rate limit exceeded. Please try again l
        debug_error_string = "{"created":"@1656941336.7988
>
[2022-07-04T13:28:56] Total uploaded: 160620 scalars, 0 te


Interrupted. View your TensorBoard at https://tensorboard.
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
    app.run(tensorboard.main, flags_parser=tensorboard.con
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py
    _run_main(main, args)
  File "/usr/local/lib/python3.7/dist-packages/absl/app.py
    sys.exit(main(argv))
  File "/usr/local/lib/python3.7/dist-packages/tensorboard
    return runner(self.flags) or 0
  File "/usr/local/lib/python3.7/dist-packages/tensorboard
    xperiment_url_callback)
  File "/usr/local/lib/python3.7/dist-packages/tensorboard
    intent.execute(server_info, channel)
KeyboardInterrupt
```

Salvo com sucesso

```
display.IFrame(
    src="https://tensorboard.dev/experiment/lZ0C6FONROaUMfjYkVyJ
    width="100%",
    height="1000px")
```




```
print(checkpoint_dir)
! ls '{checkpoint_dir}'
#! ls '/content/drive/MyDrive/disciplinasDoutorado/PCC177-2022-1
```

```
/content/drive/MyDrive/disciplinasDoutorado/PCC177-2022-1(
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
```

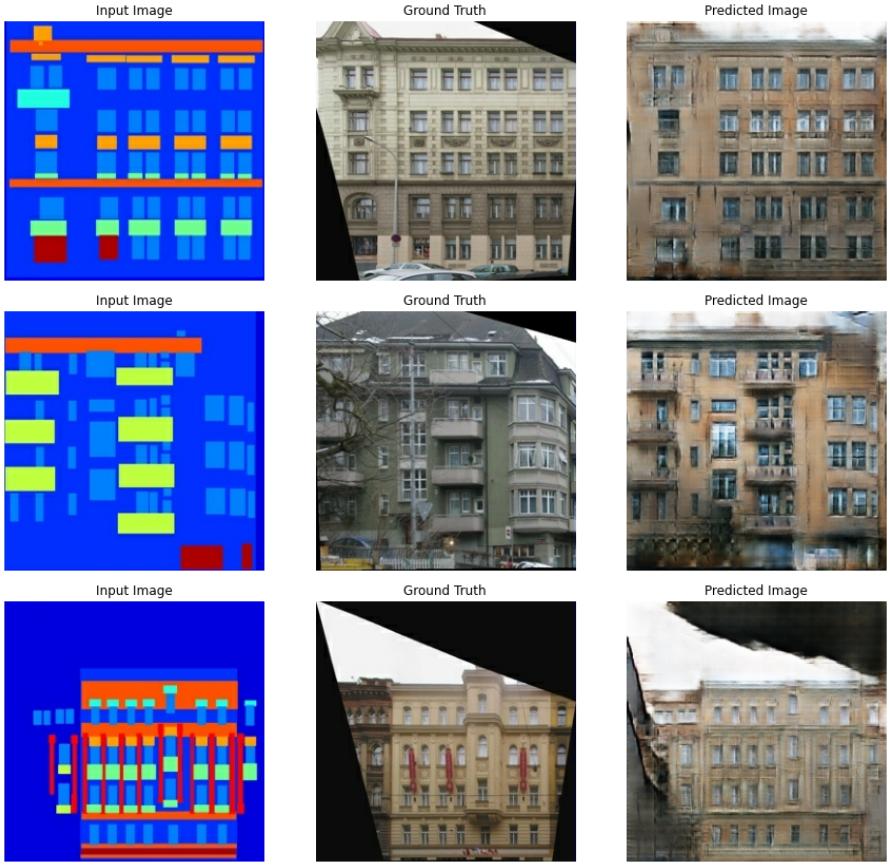
```
ckpt-4.data-00000-of-00001  ckpt-8.index
ckpt-4.index
```

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

```
<tensorflow.python.training.tracking.util.CheckpointLoadSt
 at 0x7ffabf132c90>
```

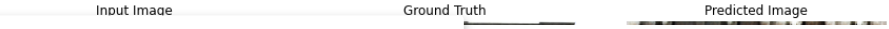
```
# Run the trained model on a few examples from the test set
for inp, tar in test_dataset.take(5):
  generate_images(generator, inp, tar)
```

Salvo com sucesso



Baseado nos exemplos do tensorflow [tutorials]

Teste com máscaras criadas (obs.: imagens anexadas)

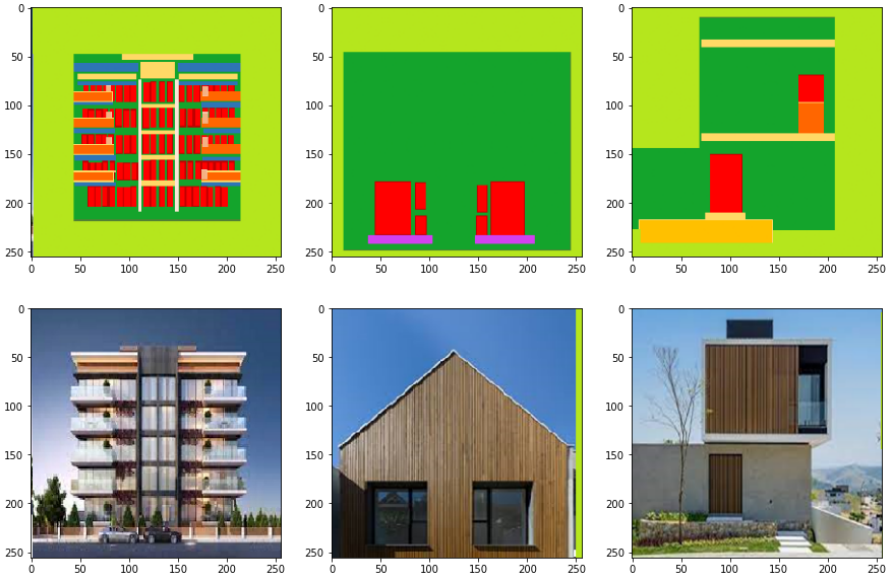


Salvo com sucesso

figsize

```
label1, facade1 = load(str('/content/drive/MyDrive/disciplinasDo
label2, facade2 = load(str('/content/drive/MyDrive/disciplinasDo
label3, facade3 = load(str('/content/drive/MyDrive/disciplinasDo

print(label1.shape, label2.shape, label3.shape)
_, axis = plt.subplots(ncols=3, nrows=2, figsize=(15, 10))
axis[0,0].imshow(label1/255.0)
axis[0,1].imshow(label2/255.0)
axis[0,2].imshow(label3/255.0)
axis[1,0].imshow(facade1/255.0)
axis[1,1].imshow(facade2/255.0)
axis[1,2].imshow(facade3/255.0)

# Para realizar o pré-processamento
label1, facade1 = load_image_test(str('/content/drive/MyDrive/di
label2, facade2 = load_image_test(str('/content/drive/MyDrive/di
label3, facade3 = load_image_test(str('/content/drive/MyDrive/di
```

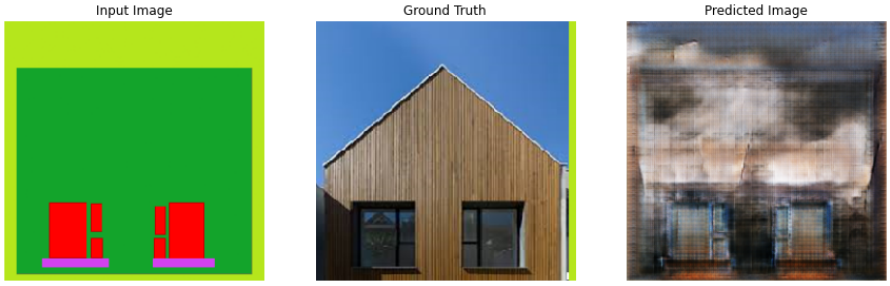
```
(256, 256, 3) (256, 256, 3) (256, 256, 3)
```



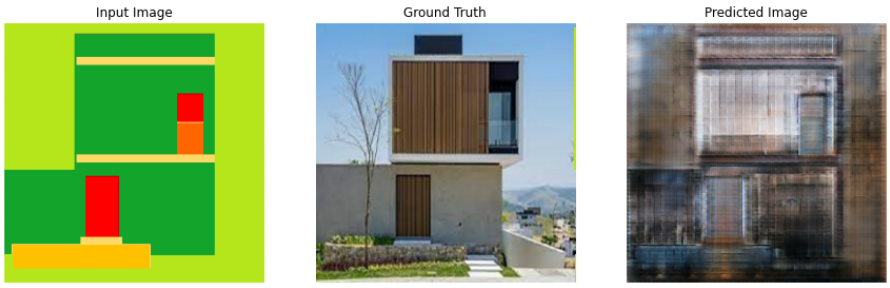
```
## Gera facade1

generate_images(generator, tf.expand_dims(label1, axis=0), tf.ex
```

Salvo com sucesso


```
## Gera facade2
generate_images(generator, tf.expand_dims(label2, axis=0), tf.ex
```




```
## Gera facade3
generate_images(generator, tf.expand_dims(label3, axis=0), tf.ex
```



Salvo com sucesso

✓ 2s conclusão: 12:28