Lab 7 - 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 : 21/10

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

Este notebook é baseado em tensorflow e Keras.

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

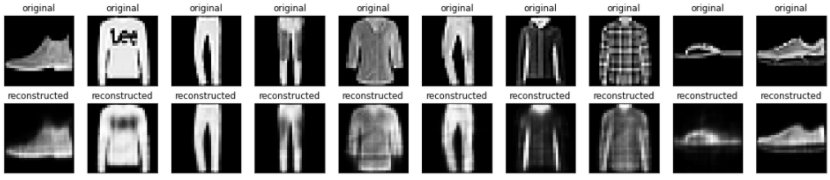
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_s
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
```

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)
```

```
Downloading data from https://storage.googleapis.c
32768/29515 [================================] -
40960/29515 [========================================
Downloading data from https://storage.googleapis.c
26427392/26421880 [==============================
26435584/26421880 [==============================
Downloading data from https://storage.googleapis.c
16384/5148 [========================================
Downloading data from https://storage.googleapis.c
4423680/4422102 [==============================] -
4431872/4422102 [==============================] -
(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'),
```

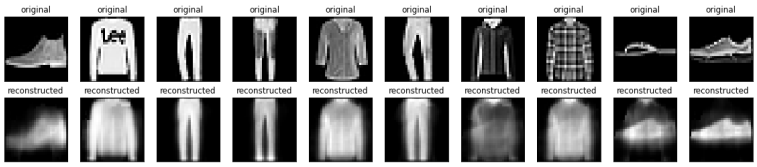
```
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

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

autoencoder = Autoencoder(latent_dim)


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


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

```
Epoch 1/10
1875/1875 [==============================] - 8s 2r
Epoch 2/10
1875/1875 [==============================] - 4s 2r
Epoch 3/10
1875/1875 [==============================] - 4s 2r
Epoch 4/10
1875/1875 [==============================] - 4s 2r
Epoch 5/10
1875/1875 [==============================] - 4s 2r
Epoch 6/10
1875/1875 [==============================] - 4s 2r
Epoch 7/10
1875/1875 [==============================] - 4s 2r
Epoch 8/10
1875/1875 [==============================] - 4s 2r
Epoch 9/10
1875/1875 [==============================] - 4s 2r
Epoch 10/10
1875/1875 [==============================] - 4s 2r
<keras.callbacks.History at 0x7f39a0336d90>
```

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])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```



ToDo : Testes (15pt)

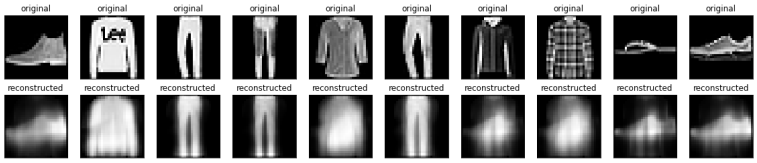
Faça testes com vetor latente de dimensões 2, 8, 16 e 64.

2 Dimensões

```
latent_dim = 2 # Definir tamanho do vetor Latente

# Criar e treinar o Auto-Encoder
autoencoder = Autoencoder(latent_dim)
autoencoder.compile(optimizer='adam', loss=losses.MeanS
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))

# Realizar Compressões e Descompressões nas imagens da
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy(

# Plotar 10 primeiros resultados
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()
```

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



8 Dimensões

```
latent_dim = 8 # Definir tamanho do vetor Latente

# Criar e treinar o Auto-Encoder
autoencoder = Autoencoder(latent_dim)
autoencoder.compile(optimizer='adam', loss=losses.MeanS
autoencoder.fit(x_train, x_train,
                epochs=10,
```

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

# Realizar Compressões e Descompressões nas imagens da
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy(

# Plotar 10 primeiros resultados
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()
```

```
Epoch 1/10
```

16 Dimensões

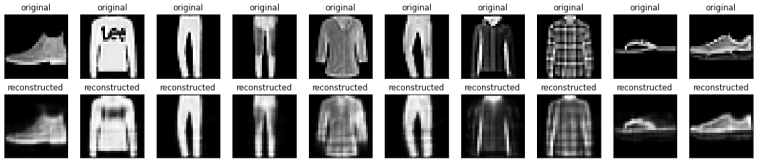
```
latent_dim = 16 # Definir tamanho do vetor Latente

# Criar e treinar o Auto-Encoder
autoencoder = Autoencoder(latent_dim)
autoencoder.compile(optimizer='adam', loss=losses.MeanS
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))

# Realizar Compressões e Descompressões nas imagens da
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy(

# Plotar 10 primeiros resultados
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()
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2r
Epoch 2/10
1875/1875 [==============================] - 4s 2r
Epoch 3/10
1875/1875 [==============================] - 4s 2r
Epoch 4/10
1875/1875 [==============================] - 4s 2r
Epoch 5/10
1875/1875 [==============================] - 4s 2r
Epoch 6/10
1875/1875 [==============================] - 4s 2r
Epoch 7/10
1875/1875 [==============================] - 4s 2r
```

64 Dimensões

```
Epoch 9/10

latent_dim = 64 # Definir tamanho do vetor Latente

# Criar e treinar o Auto-Encoder
autoencoder = Autoencoder(latent_dim)
autoencoder.compile(optimizer='adam', loss=losses.MeanS
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))

# Realizar Compressões e Descompressões nas imagens da
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy(

# Plotar 10 primeiros resultados
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()
```

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



ToDo : Responda (15pt)

Escreva suas conclusões sobre os testes executados

Quando é aumentado o número de dimensões do vetor latente, a imagem decodificada fica mais parecida com a imagem original.

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/
raw_data = dataframe.values
dataframe.head()
```

	0	1	2	3	
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338

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 = trai
    data, labels, test_size=0.2, random_state=21
)
```

Normaliza entre `[0,1]`.

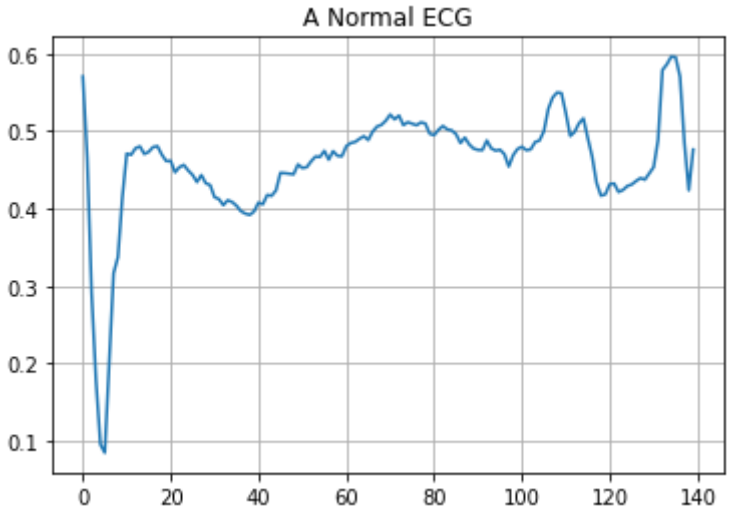
```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) / (max_val - min_va
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.

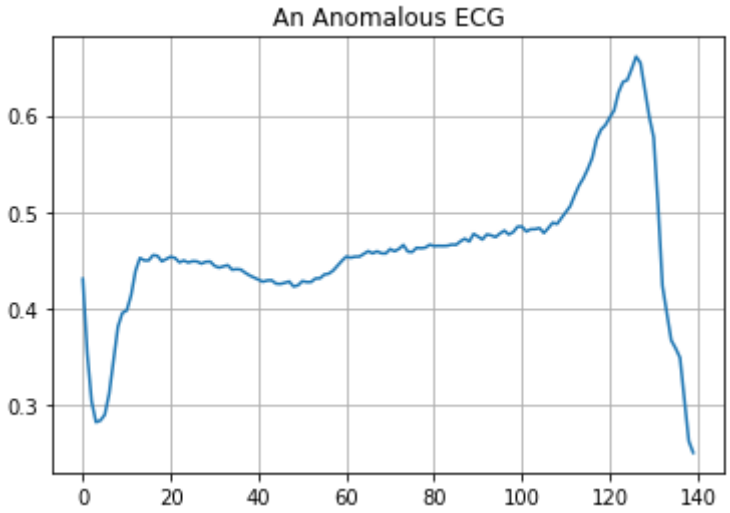
```
plt.grid()
plt.plot(np.arange(140), normal_train_data[0])
plt.title("A Normal ECG")
plt.show()
```



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)

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

```
# Importações
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import Input


class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    #self.encoder = tf.keras.Sequential([
      # Todo: crie o encoder, com um gargalo
    self.encoder = tf.keras.Sequential([
                                        Conv1D(filters=
                                        Conv1D(filters=
                                        layers.Flatten(
                                        layers.Dense(52
                                        layers.Dense(1,
                                        layers.Dense(52
    #self.decoder = tf.keras.Sequential([
      # Todo: crie as camadas do decoder
    self.decoder = tf.keras.Sequential([layers.Flatten(
                                        layers.Dense(52
                                        layers.Dense(14

  def call(self, x):
```

```
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder2 = AnomalyDetector()


class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    #self.encoder = tf.keras.Sequential([
      # Todo: crie o encoder, com um gargalo
    self.encoder = tf.keras.Sequential([layers.Flatten(
                                        layers.Dense(52
                                        layers.Dense(1,
                                        layers.Dense(52
    #self.decoder = tf.keras.Sequential([
      # Todo: crie as camadas do decoder
    self.decoder = tf.keras.Sequential([layers.Dense(14
  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = AnomalyDetector()


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


autoencoder2.compile(optimizer='adam', loss='mae')
```

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

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

```
Epoch 1/20
5/5 [==============================] - 1s 36ms/ste
Epoch 2/20
5/5 [==============================] - 0s 8ms/step
Epoch 3/20
5/5 [==============================] - 0s 8ms/step
Epoch 4/20
5/5 [==============================] - 0s 9ms/step
Epoch 5/20
5/5 [==============================] - 0s 9ms/step
Epoch 6/20
5/5 [==============================] - 0s 8ms/step
Epoch 7/20
```

```
5/5 [==============================] - 0s 8ms/step
Epoch 8/20
5/5 [==============================] - 0s 9ms/step
Epoch 9/20
5/5 [==============================] - 0s 8ms/step
Epoch 10/20
5/5 [==============================] - 0s 8ms/step
Epoch 11/20
5/5 [==============================] - 0s 9ms/step
Epoch 12/20
5/5 [==============================] - 0s 8ms/step
Epoch 13/20
5/5 [==============================] - 0s 12ms/ste
Epoch 14/20
5/5 [==============================] - 0s 8ms/step
Epoch 15/20
5/5 [==============================] - 0s 8ms/step
Epoch 16/20
5/5 [==============================] - 0s 8ms/step
Epoch 17/20
5/5 [==============================] - 0s 8ms/step
Epoch 18/20
5/5 [==============================] - 0s 8ms/step
Epoch 19/20
5/5 [==============================] - 0s 8ms/step
Epoch 20/20
5/5 [==============================] - 0s 11ms/ste
```

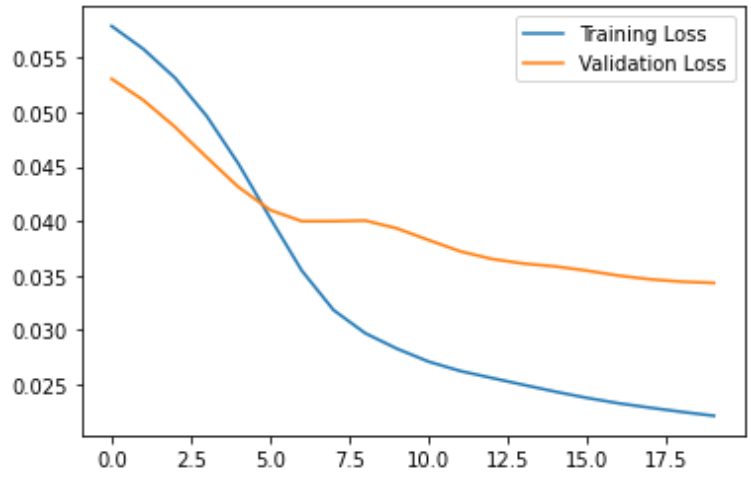
```
history2 = autoencoder2.fit(np.expand_dims(normal_train
          epochs=20,
          batch_size=512,
          validation_data=(np.expand_dims(test_data, ax
          shuffle=True)
```

```
Epoch 1/20
5/5 [==============================] - 9s 89ms/ste
Epoch 2/20
5/5 [==============================] - 0s 22ms/ste
Epoch 3/20
5/5 [==============================] - 0s 21ms/ste
Epoch 4/20
5/5 [==============================] - 0s 21ms/ste
Epoch 5/20
5/5 [==============================] - 0s 18ms/ste
Epoch 6/20
5/5 [==============================] - 0s 18ms/ste
Epoch 7/20
5/5 [==============================] - 0s 18ms/ste
Epoch 8/20
5/5 [==============================] - 0s 18ms/ste
Epoch 9/20
5/5 [==============================] - 0s 20ms/ste
Epoch 10/20
5/5 [==============================] - 0s 17ms/ste
Epoch 11/20
5/5 [==============================] - 0s 19ms/ste
```

```
Epoch 12/20
5/5 [==============================] - 0s 17ms/ste
Epoch 13/20
5/5 [==============================] - 0s 17ms/ste
Epoch 14/20
5/5 [==============================] - 0s 17ms/ste
Epoch 15/20
5/5 [==============================] - 0s 17ms/ste
Epoch 16/20
5/5 [==============================] - 0s 17ms/ste
Epoch 17/20
5/5 [==============================] - 0s 19ms/ste
Epoch 18/20
5/5 [==============================] - 0s 18ms/ste
Epoch 19/20
5/5 [==============================] - 0s 18ms/ste
Epoch 20/20
5/5 [==============================] - 0s 17ms/ste
```

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

```
<matplotlib.legend.Legend at 0x7f392062a690>
```

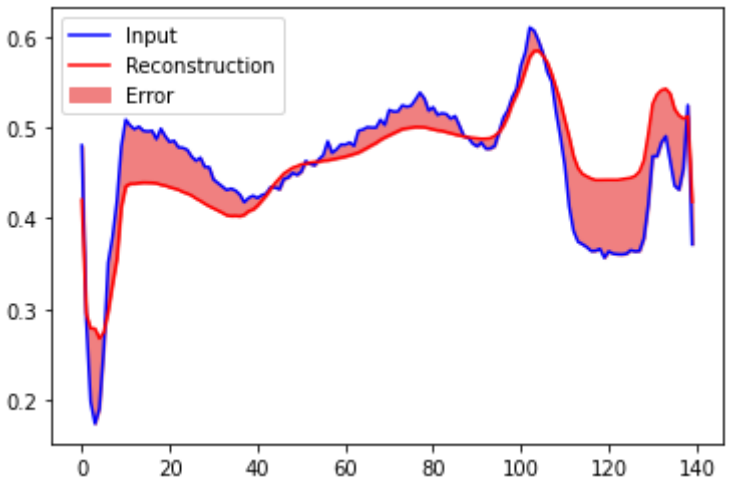



```
plt.plot(history2.history["loss"], label="Training Loss
plt.plot(history2.history["val_loss"], label="Validatio
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f3920502dd0>
```

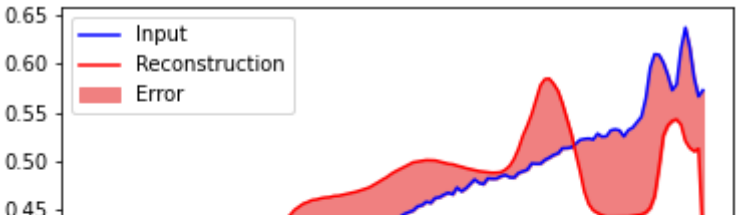
Você vai considerar um batimento como anômalo se ele divergir mais que um desvio padrão das amostras normais. Primeiro, vamos plotar um batimento normal a partir da base de treino e sua reconstrução. Assim, poderemos calcular o erro de re-construção.

```
encoded_data = autoencoder.encoder(normal_test_data).nu
decoded_data = autoencoder.decoder(encoded_data).numpy(

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

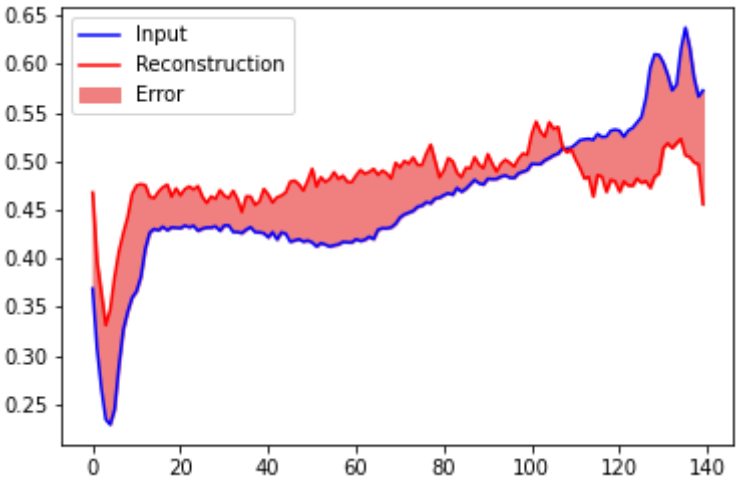


Vamos fazer o mesmo para um batimento anômalo.

```
encoded_data = autoencoder.encoder(anomalous_test_data)
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], anoma
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```




```
encoded_data2 = autoencoder2.encoder(np.expand_dims(ano
decoded_data2 = autoencoder2.decoder(encoded_data).nump

plt.plot(anomalous_test_data[0], 'b')
plt.plot(decoded_data2[0], 'r')
plt.fill_between(np.arange(140), decoded_data2[0], anom
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```

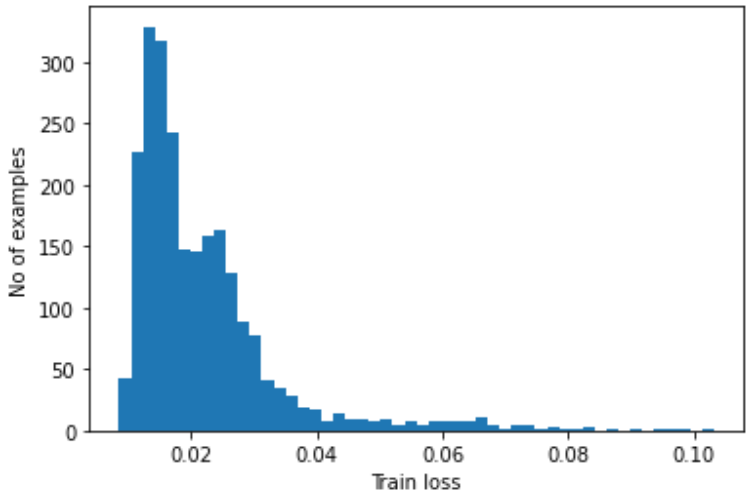


Detectando as anomalias

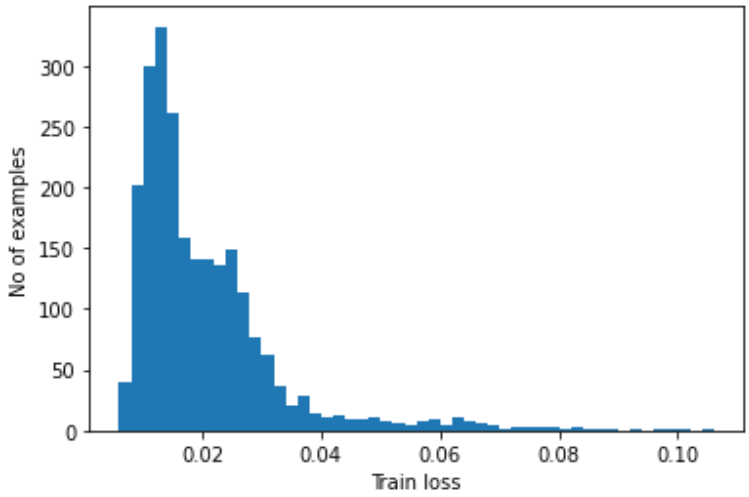
Vamos detectar as anomalis se o erro de reconstrução for maior que um limiar. Aqui, vamos calcular o erro médio para os exemp;los normais do treino e depois, classificar os anormais do teste, que tenha erro de reconstrução maior que um desvio padrão.

Plota erro de reconstrução de batimentos normais do treino

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

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




```
reconstructions2 = autoencoder2.predict(np.expand_dims(
train_loss2 = tf.keras.losses.mae(reconstructions2, nor

plt.hist(train_loss2[None,:], bins=50)
plt.xlabel("Train loss")
plt.ylabel("No of examples")
plt.show()
```



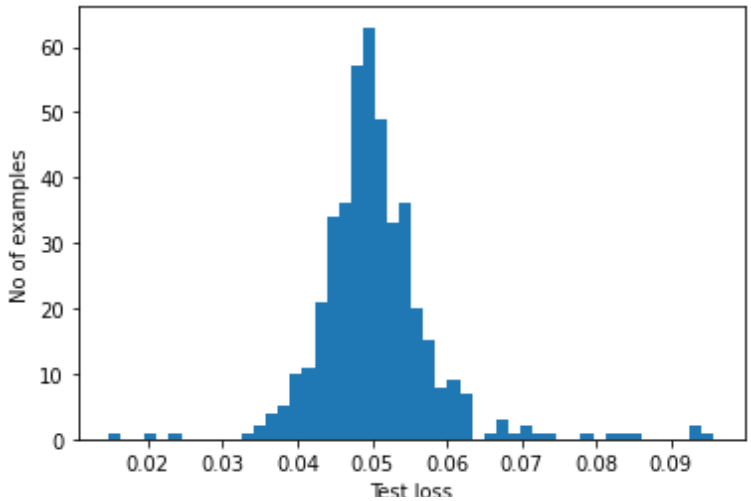
Escolha do limiar.

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

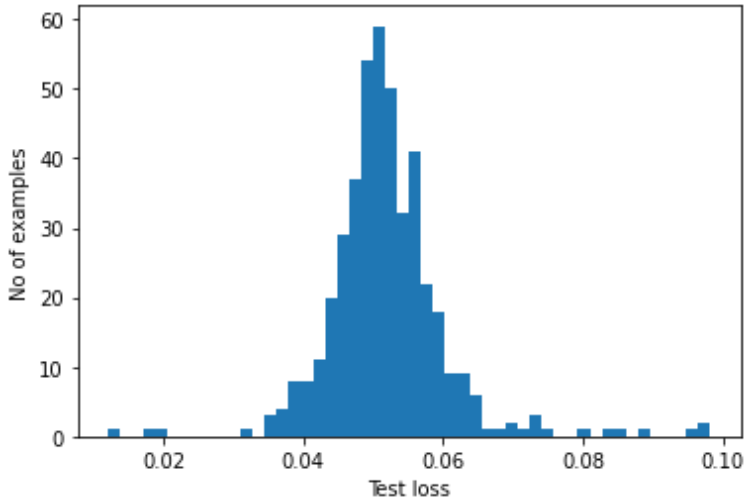
```
Threshold:  0.034146238
```

```
reconstructions = autoencoder.predict(anomalous_test_da
test_loss = tf.keras.losses.mae(reconstructions, anomal

plt.hist(test_loss[None, :], bins=50)
plt.xlabel("Test loss")
```

```
plt.ylabel("No of examples")
plt.show()
```




```
reconstructions2 = autoencoder2.predict(np.expand_dims(
test_loss2 = tf.keras.losses.mae(reconstructions2, anom

plt.hist(test_loss2[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, p
  print("Precision = {}".format(precision_score(labels,
  print("Recall = {}".format(recall_score(labels, predi
```

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

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

```
Accuracy = 0.945
Precision = 0.9922027290448343
Recall = 0.9089285714285714
```

```
preds = predict(autoencoder2, np.expand_dims(test_data,

predito = []
for i in range(preds.shape[0]):
  predito.append(bool(preds[i][0]))

print_stats(predito, test_labels)
```

```
Accuracy = 0.948
Precision = 0.9922480620155039
Recall = 0.9142857142857143
```

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 representando 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)

Importações

```
import tensorflow as tf
```

```
import os
import pathlib
import time
import datetime

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

Carregar o conjunto de dados

```
dataset_name = "facades"


_URL = f'http://efrosgans.eecs.berkeley.edu/pix2pix/dat

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.berke
30171136/30168306 [==============================
30179328/30168306 [==============================
```

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

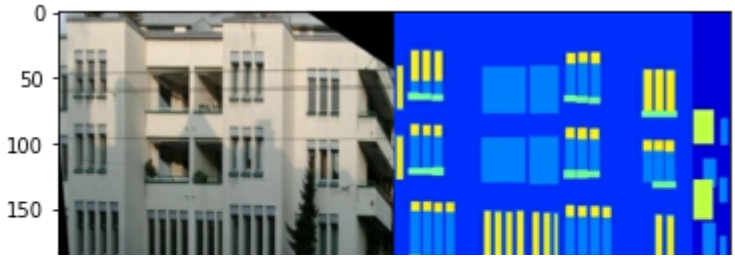
```
[PosixPath('/root/.keras/datasets/fashion-
mnist'),

PosixPath('/root/.keras/datasets/facades.tar.gz'),

 PosixPath('/root/.keras/datasets/facades')]
```

```
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)
```

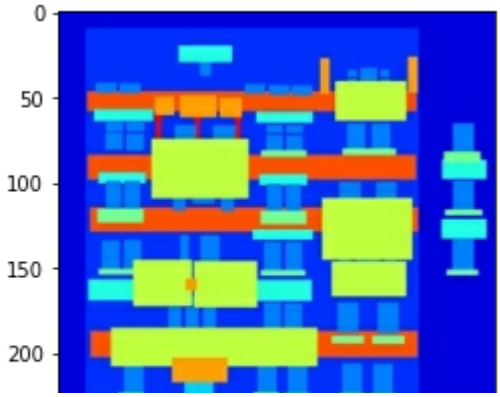
```
<matplotlib.image.AxesImage at 0x7f38fe2feb10>
```



Separar imagens reais das fachadas

```
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)
```

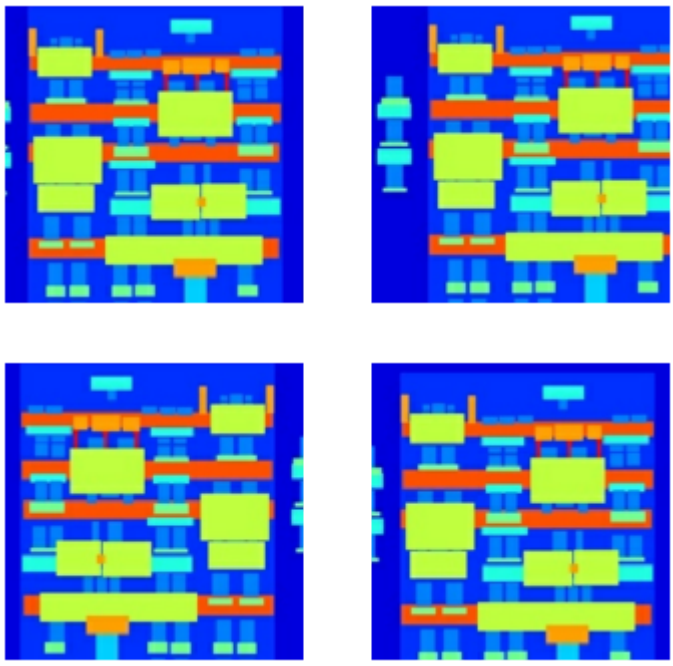
```
<matplotlib.image.AxesImage at 0x7f38b6ba3450>
```




```
# The facade training set consist of 400 images
BUFFER_SIZE = 400
# The batch size of 1 produced better results for the U
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, w
                                method=tf.image.ResizeM
  real_image = tf.image.resize(real_image, [height, wid
                               method=tf.image.ResizeMe

  return input_image, real_image


def random_crop(input_image, real_image):
  stacked_image = tf.stack([input_image, real_image], a
  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_im

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

  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()
```



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

  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_im
                                   IMG_HEIGHT, IMG_WIDT
  input_image, real_image = normalize(input_image, real

  return input_image, real_image
```

Criar Pipeline

```
train_dataset = tf.data.Dataset.list_files(str(PATH / '
train_dataset = train_dataset.map(load_image_train,
```

```
                                    num_parallel_calls=tf
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.batch(BATCH_SIZE)


try:
  test_dataset = tf.data.Dataset.list_files(str(PATH /
except tf.errors.InvalidArgumentError:
  test_dataset = tf.data.Dataset.list_files(str(PATH /
test_dataset = test_dataset.map(load_image_test)
test_dataset = test_dataset.batch(BATCH_SIZE)
```

Construir Gerador

```
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,
                             kernel_initializer=initial

  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)
```

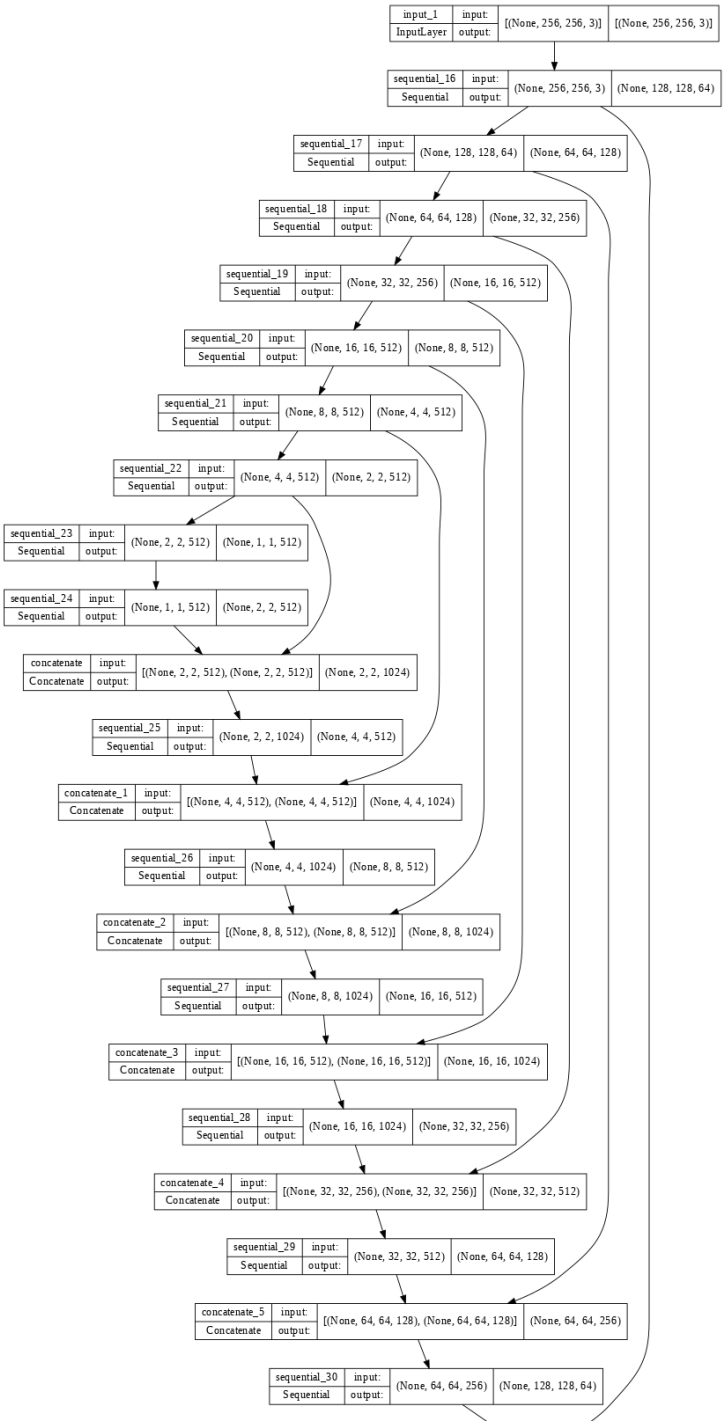
```
def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
    tf.keras.layers.Conv2DTranspose(filters, size, stri
                                    padding='same',
                                    kernel_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)
```

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

  down_stack = [
    downsample(64, 4, apply_batchnorm=False),  # (batch
    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_siz
    upsample(512, 4, apply_dropout=True),  # (batch_siz
    upsample(512, 4, apply_dropout=True),  # (batch_siz
    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_CHANNEL
                                         strides=2,
                                         padding='same'
                                         kernel_initial
                                         activation='ta

  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,
```



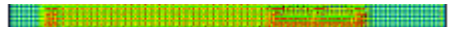

```
gen_output = generator(inp[tf.newaxis, ...], training=F
plt.imshow(gen_output[0, ...])
```

```
WARNING:matplotlib.image:Clipping input data to th
<matplotlib.image.AxesImage at 0x7f39362a97d0>
```

Definir Perda do Gerador

```
LAMBDA = 100
```

```
loss_object = tf.keras.losses.BinaryCrossentropy(from_l
```

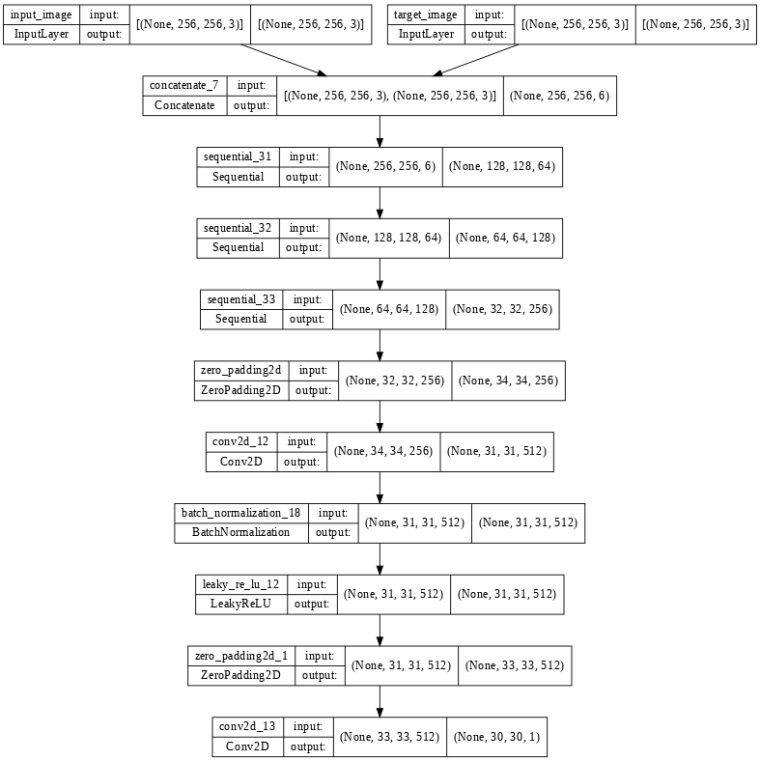


```
def generator_loss(disc_generated_output, gen_output, t
  gan_loss = loss_object(tf.ones_like(disc_generated_ou

  # 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
```

Contruir Discriminador

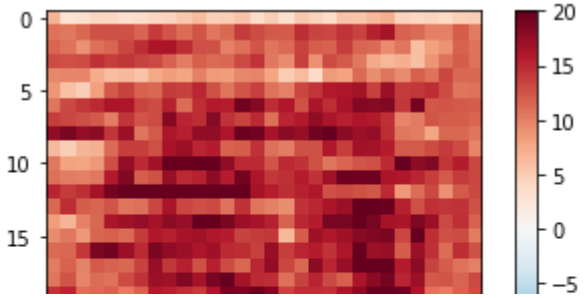
```
def Discriminator():
  initializer = tf.random_normal_initializer(0., 0.02)

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

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

  down1 = downsample(64, 4, False)(x)  # (batch_size, 1
  down2 = downsample(128, 4)(down1)  # (batch_size, 64,
  down3 = downsample(256, 4)(down2)  # (batch_size, 32,

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

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

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

  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_rel

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

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


discriminator = Discriminator()
```

```
discriminator = Discriminator()
tf.keras.utils.plot_model(discriminator, show_shapes=Tr
```




```
disc_out = discriminator([inp[tf.newaxis, ...], gen_out
plt.imshow(disc_out[0, ..., -1], vmin=-20, vmax=20, cma
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f38b6ad93d0>
```



Definir a perda do discriminador


```
def discriminator_loss(disc_real_output, disc_generated
  real_loss = loss_object(tf.ones_like(disc_real_output

  generated_loss = loss_object(tf.zeros_like(disc_gener

  total_disc_loss = real_loss + generated_loss

  return total_disc_loss
```

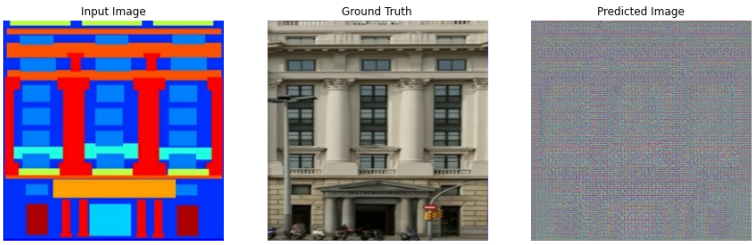
Definir otimizadores e um protetor de ponto de verificação

```
generator_optimizer = tf.keras.optimizers.Adam(2e-4, be
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4


checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt"
checkpoint = tf.train.Checkpoint(generator_optimizer=ge
                                 discriminator_optimize
                                 generator=generator,
                                 discriminator=discrimi
```

Gerar Imagens

```
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 Im

  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 p
    plt.imshow(display_list[i] * 0.5 + 0.5)
    plt.axis('off')
  plt.show()
```

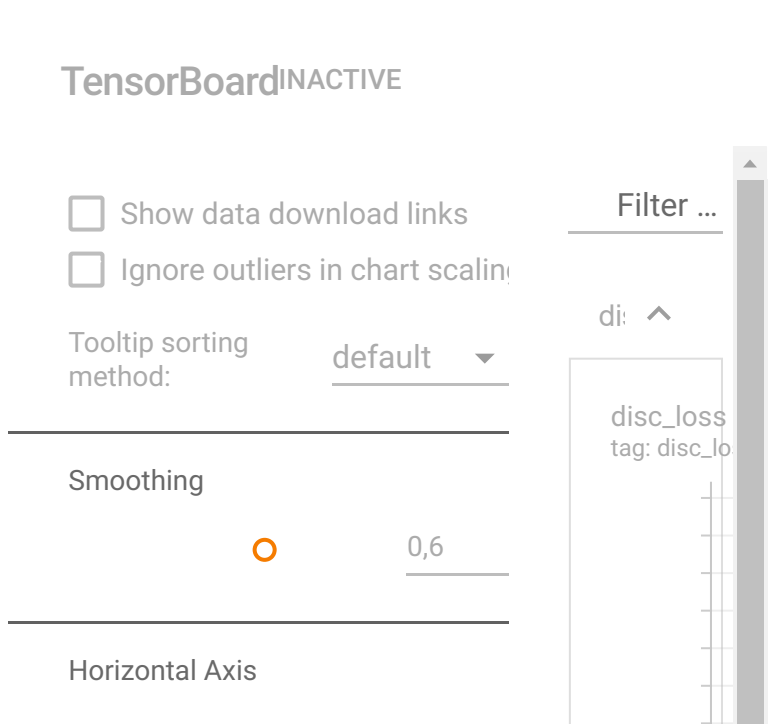
```
for example_input, example_target in test_dataset.take(
  generate_images(generator, example_input, example_tar
```



Treinamento

```
log_dir="logs/"

summary_writer = tf.summary.create_file_writer(
  log_dir + "fit/" + datetime.datetime.now().strftime("


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

    disc_real_output = discriminator([input_image, targ
    disc_generated_output = discriminator([input_image,

    gen_total_loss, gen_gan_loss, gen_l1_loss = generat
    disc_loss = discriminator_loss(disc_real_output, di

  generator_gradients = gen_tape.gradient(gen_total_los
                                          generator.tra
  discriminator_gradients = disc_tape.gradient(disc_los
                                               discrimi

  generator_optimizer.apply_gradients(zip(generator_gra
                                          generator.tra
  discriminator_optimizer.apply_gradients(zip(discrimin
                                              discrimin

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

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

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

      if step != 0:
        print(f'Time taken for 1000 steps: {time.time()

      start = time.time()

      generate_images(generator, example_input, example
      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)


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

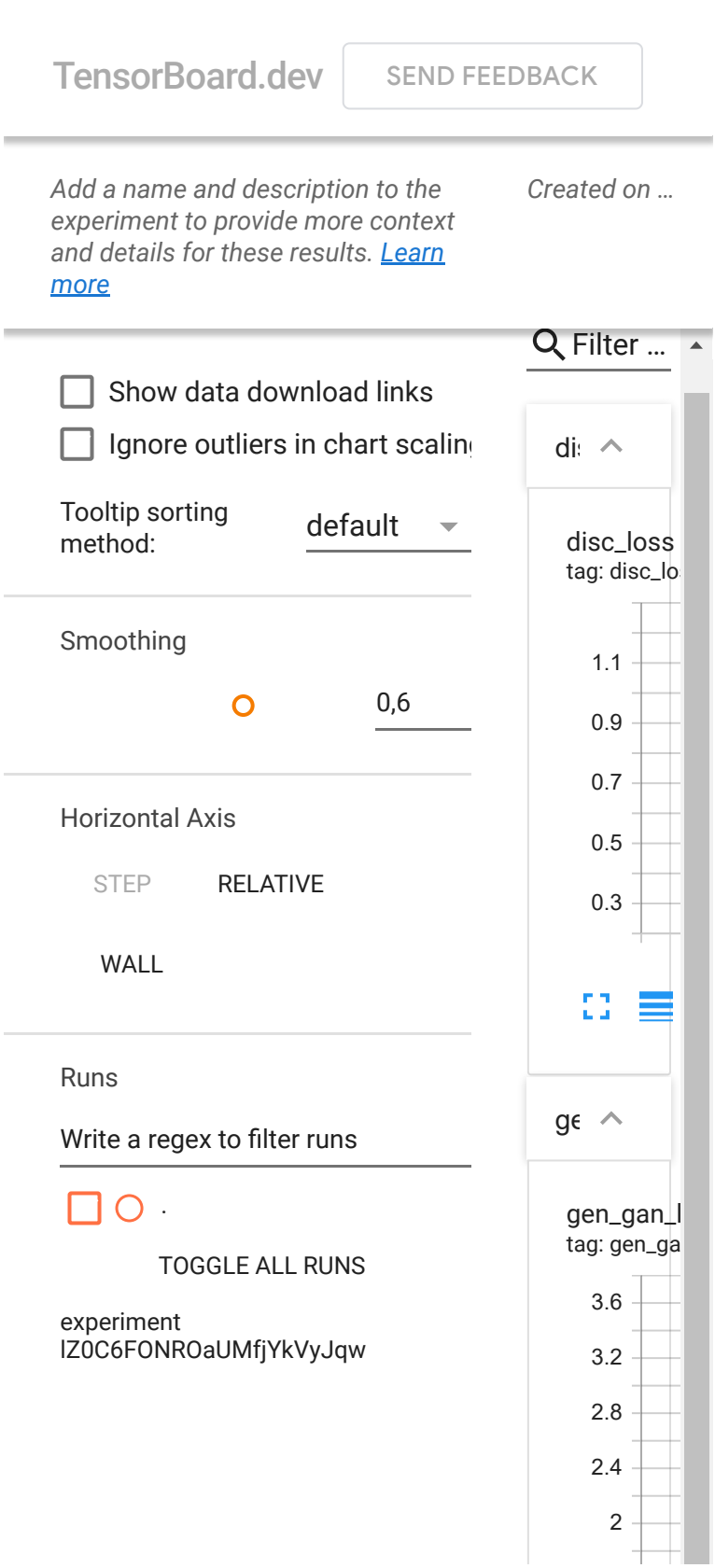



```
fit(train_dataset, test_dataset, steps=2000)
```

```
Time taken for 1000 steps: 98.10 sec
```


```
Step: 1k
.................................................
```

```
display.IFrame(
    src="https://tensorboard.dev/experiment/lZ0C6FONROa
    width="100%",
    height="1000px")
```



Restaure o ponto de verificação mais recente e teste a rede

```
ls {checkpoint_dir}
```

```
checkpoint                  ckpt-3.index
ckpt-1.data-00000-of-00001  ckpt-4.data-00000-of-(
ckpt-1.index                ckpt-4.index
ckpt-2.data-00000-of-00001  ckpt-5.data-00000-of-(
```

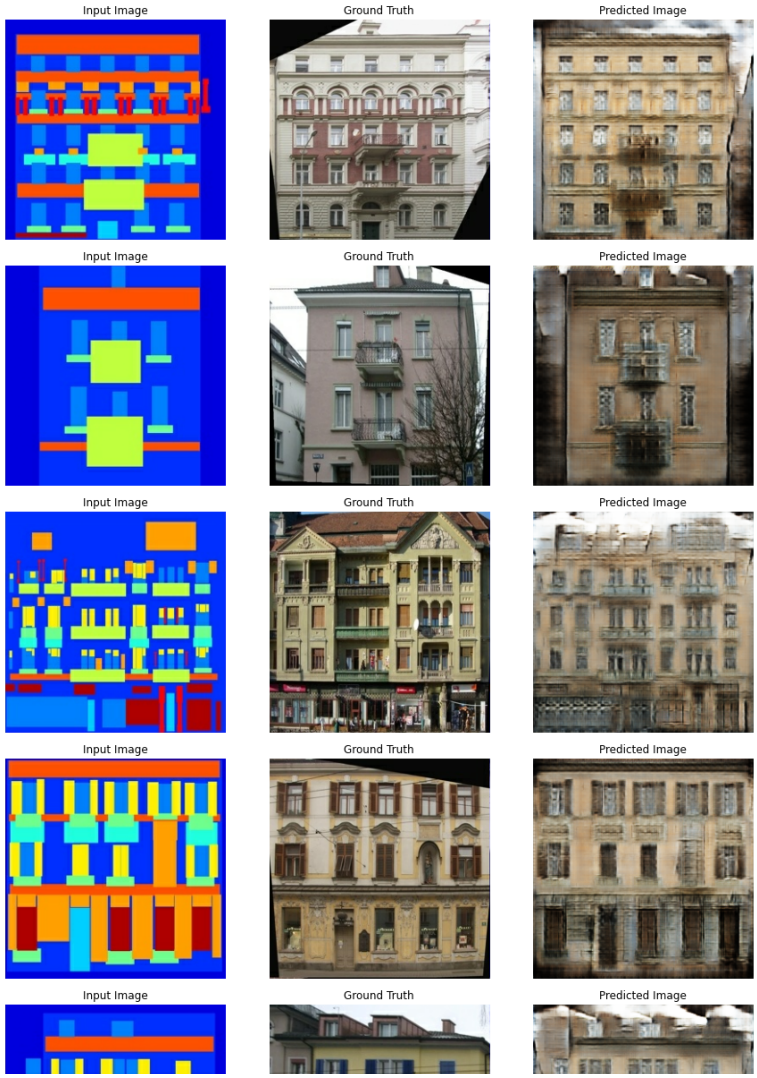
```
ckpt-2.index                ckpt-5.index
ckpt-3.data-00000-of-00001
```

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

```
<tensorflow.python.training.tracking.util.Checkpo:
 at 0x7f38b6b134d0>
```

Gerar algumas imagens usando o conjunto de teste

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



Implementar

```
# ToDO : Criar 3 máscaras e gerar 3 saídas com a pix2pi
```

Baseado nos exemplos do tensorflow tutorials

```
# Importações
import numpy as np
import os
import cv2
from google.colab.patches import cv2_imshow
```

```
# Ligar ao Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attem
```

```
#Ler Imagens (Criei pasta prédios no drive)
root_dir = '/content/drive/My Drive/predios/'
```

```
imagens = {}
for i in range(1,4):
  input_image = cv2.imread("/content/drive/My Drive/pre
  real_image = cv2.imread("/content/drive/My Drive/pred

  input_image = cv2.resize(input_image, dsize=(256,256)
  real_image = cv2.resize(real_image, dsize=(256,256))

  input_image = np.asarray(input_image)*1.0
  real_image = np.asarray(real_image)*1.0
  """  print(input_image)
  cv2_imshow(input_image)
  cv2_imshow(real_image) """
  #input_image, real_image = random_jitter(input_image/
  #cv2_imshow( np.asarray(input_image))
  #input_image, real_image = normalize(input_image, rea
  #cv2_imshow( np.asarray(input_image))
  input_image = np.expand_dims(input_image, axis=0)
  real_image = np.expand_dims(real_image, axis=0)
  imagens["Predio"+str(i)] = [input_image/255, real_ima


cv2_imshow(imagens["Predio1"][0][0])
cv2_imshow(imagens["Predio1"][1][0])
```

```
# Run the trained model on a few examples from the test
for predio in imagens.keys():
```

```
generate_images(generator,imagens[predio][0], imagens
```