

# Inteligência Artificial



## Deep Learning

Professor. Dr. Rogério de Oliveira

TURMA 01A – MATRÍCULA 92104843 Fernando Antonio Carvalho Pessoa

Tarefa da trilha 4: Modelos Sequenciais e Classificação com Keras TensorFlow

## Introdução

Nesta tarefa devemos implementar um modelo de classificação binária ou multiclasse para um conjunto de dados TensorFlow e O Keras.

## Dataset

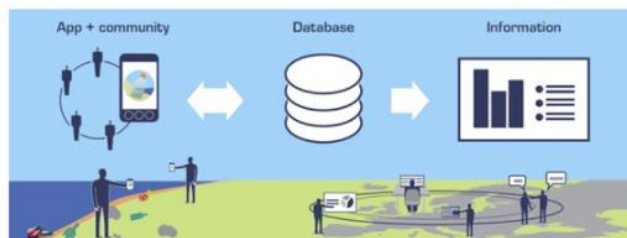
O dataset escolhido é o MLW\_Data, este dataset fornecido pela Agência Europeia de Ambiente, é criado a partir de um aplicativo denominado LitterWatch, que é utilizado nas comunidades costeiras da Europa para identificar o lixo jogado ao oceano.

## Fonte da base

<https://www.eea.europa.eu/data-and-maps/data/marine-litter>

## Visualização por TABLEAU

<https://www.eea.europa.eu/themes/water/europes-seas-and-coasts/assessments/marine-litterwatch/data-and-results/marine-litterwatch-data-viewer/marine-litterwatch-data-viewer>



## Objetivo do modelo

Tendo o levantamento do lixo, por tipo e quantidade, localizado em cada comunidade, nosso objetivo é treinar o modelo para identificar e categorizar o local como POLUIDO ou LIMPO

## Importando Bibliotecas

```
[1] 1 import pandas as pd
    2 from IPython.display import display
    3 import seaborn as sns
    4 import numpy as np
    5 from tensorflow import keras
    6 from tensorflow.keras import layers
    7 import tensorflow as tf
```

## Importando datasets

Carregando os dados coletados no aplicativo e a tabela de categorias de lixo

### Base de dados do agrupamento por tipo de lixo

```
[2] 1 # carregando dados de agrupamento para o Google Colab
2   from google.colab import files
3   uploaded = files.upload()
```

Escolher arquivos MLW\_Meta.csv

• MLW\_Meta.csv(application/vnd.ms-excel) - 6826 bytes, last modified: 06/09/2021 - 100% done  
Saving MLW\_Meta.csv to MLW\_Meta.csv

```
[3] 1 # Carregando o arquivo CSV em dataframe
2   Titulos = pd.read_csv('/content/MLW_Meta.csv', engine= 'python', sep = ';', encoding='utf-8')
```

```
[4] 1 #Análise da quantidade de colunas e linhas
2   Titulos.shape
```

(164, 3)

```
[5] 1 Titulos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   generalcode  164 non-null    object
1   category     164 non-null    object
2   generalname  164 non-null    object
dtypes: object(3)
memory usage: 4.0+ KB
```

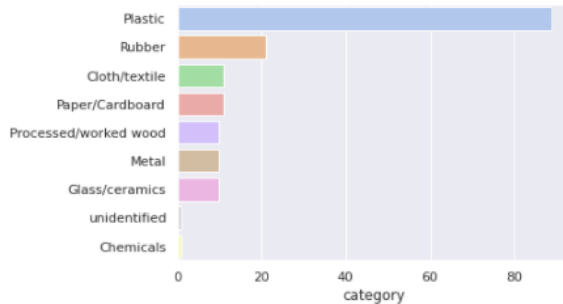
```
[6] 1 Titulos.head(10)
```

	generalcode	category	generalname
0	G1	Plastic	4/6-pack yokes, six-pack rings
1	G3	Plastic	Shopping Bags incl. pieces
2	G4	Plastic	Small plastic bags, e.g. freezer bags incl. pi...
3	G5	Plastic	Plastic bags collective role what remains from...
4	G7	Plastic	Drink bottles <=0.5l
5	G8	Plastic	Drink bottles >0.5l
6	G9	Plastic	Cleaner bottles & containers
7	G10	Plastic	Food containers incl. fast food containers
8	G11	Plastic	Beach use related cosmetic bottles and contain...
9	G12	Plastic	Other cosmetics bottles & containers

```
[7] 1 Titulos.groupby('category')['category'].count()
```

```
category
Chemicals          1
Cloth/textile      11
Glass/ceramics     10
Metal              21
Paper/Cardboard    10
Plastic            89
Processed/worked wood 11
Rubber             10
unidentified        1
Name: category, dtype: int64
```

```
[8] 1 sns.set(style="darkgrid")
2 categoria = Titulos['category'].unique()
3 cont = Titulos['category'].value_counts()
4 sns.barplot(x=cont,y=categoria, palette='pastel',orient='h');
```



## Base de dados da pesquisa

```
[9] 1 # carregando o arquivo dados para o Google Colab
2 from google.colab import files
3 uploaded = files.upload()
```

Escolher arquivos MLW\_Data.csv

- MLW\_Data.csv(application/vnd.ms-excel) - 100156 bytes, last modified: 06/09/2021 - 100% done
- Saving MLW\_Data.csv to MLW\_Data.csv

```
[10] 1 data = pd.read_csv('/content/MLW_Data.csv', engine= 'python', sep = ';', encoding='latin-1')
```

```
[11] 1 data.shape
```

(254, 178)

```
[12] 1 data.head(5)
```

	CommunityName	BeachName	BeachCountrycode	BeachRegionalSea	BeachLength_m	BeachLocation	BeachType	EventDate	EventType	NatRef	lon_x1	lat_y1
0	gBqsPxAZ	Neum	BA	Mediterranean Sea	1551.0	Urban	Other (mixed)	20160424.0	Cleanup	NaN	17.619.790.849.047.900	4.291.792.327.340.360
1	gBqsPxAZ	Ponton	BA	NaN	86.0	Urban	Other (mixed)	20160519.0	Cleanup	NaN	1.910.840.665.135.820	44.368.655.173.785.800
2	Surfrider Foundation Europe	Blakenberg beach	BE	North-east Atlantic Ocean	82.0	Urban	Sandy	20160812.0	Cleanup	NaN	29.747.061.142.684.500	51.255.998.063.635.000
3	Perseus	Alepu	BG	Black Sea	105.0	Rural	Sandy	20160317.0	Cleanup	NaN	2.770.921.843.173.200	42.367.082.331.608.300
4	gBqsPxAZ	alepu	BG	Black Sea	2779.0	Urban	Sandy	20160313.0	Cleanup	NaN	27.722.467.974.911.600	4.234.864.091.261.680

5 rows × 178 columns

## Pré processamento da base

O objetivo desta etapa é tratar, preparar e montar os dados coletados em uma base para aplicação do algoritmo Keras - TensorFlow

```
1 #Verifica valores NAN
2 data.isnull().sum()
```

```
CommunityName      1
BeachName           1
BeachCountrycode    1
BeachRegionalSea    2
BeachLength_m       1
...
G208                147
G210                216
G211                208
G213                235
CLASSE              0
Length: 178, dtype: int64
```

[14] 1 data.head()

	G1	G3	G4	G5	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G21	G22	G23	G24	G25	G26	G27	G28	...	G164	G165	G166	G167	G171	G172	G174	
	NaN	37.0	15.0	NaN	56.0	17.0	NaN	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	90.0	NaN	NaN	NaN	NaN	NaN	895.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
	NaN	10.0	3.0	NaN	148.0	144.0	8.0	33.0	NaN	6.0	NaN	6.0	NaN	NaN	1.0	NaN	NaN	350.0	NaN	NaN	NaN	NaN	NaN	2.0	24.0	NaN	...	NaN	NaN	NaN	NaN	44.0	32.0	3.0
	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.0	NaN	NaN	NaN	NaN	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
	NaN	NaN	2.0	NaN	5.0	1.0	NaN	2.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	41.0	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	NaN	26.0	NaN	NaN	14.0	16.0	4.0	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	36.0	4.0	32.0	36.0	2.0	1.0	20.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

```
1 #Substitui os valores NaN por zero
2 dataT = data.fillna(0)
3 dataT.isnull().sum()
```

CommunityName	0
BeachName	0
BeachCountrycode	0
BeachRegionalSea	0
BeachLength_m	0
..	
G208	0
G210	0
G211	0
G213	0
CLASSE	0
Length: 178, dtype: int64	

[16] 1 dataT.head()

	G1	G3	G4	G5	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G21	G22	G23	G24	G25	G26	G27	G28	...	G164	G165	G166	G167	G171	G172	G174	G175	G176	
	0.0	37.0	15.0	0.0	56.0	17.0	0.0	14.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	90.0	0.0	0.0	0.0	0.0	0.0	895.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
	0.0	10.0	3.0	0.0	148.0	144.0	8.0	33.0	0.0	6.0	0.0	6.0	0.0	0.0	1.0	0.0	0.0	350.0	0.0	0.0	0.0	0.0	0.0	2.0	24.0	0.0	...	0.0	0.0	0.0	0.0	44.0	32.0	3.0	4.0	0.0
	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
	0.0	0.0	2.0	0.0	5.0	1.0	0.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	41.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
	0.0	26.0	0.0	0.0	14.0	16.0	4.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	36.0	4.0	32.0	36.0	2.0	1.0	20.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0	0.0	

## Preparando os Dados

As colunas que precisamos são os indicadores de poluição (coluna G1 a G213) e a coluna de atributo Poluido = 1 não poluido = 0

Também necessitamos substituir as quantidades das colunas G1 a g213 por atributo do tipo de poluente encontrado=1 ou não encontrado=0

### Eliminando colunas

```
[17] 1 dataT1 = dataT.drop(columns=['CommunityName','BeachName','BeachCountrycode','BeachRegionalSea','BeachLength_m','BeachLocation','BeachType','EventDate','EventType','NatRef','lon_x1',
2 dataT1.head()

```

	G1	G3	G4	G5	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G21	G22	G23	G24	G25	G26	G27	G28	G29	G30	G31	G32	G33	G34	G35	G36	G37	G40	G41
0	0.0	37.0	15.0	0.0	56.0	17.0	0.0	14.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	90.0	0.0	0.0	0.0	0.0	0.0	895.0	0.0	0.0	28.0	0.0	0.0	0.0	70.0	45.0	0.0	0.0	0.0	0.0
1	0.0	10.0	3.0	0.0	148.0	144.0	8.0	33.0	0.0	6.0	0.0	6.0	0.0	0.0	1.0	0.0	0.0	350.0	0.0	0.0	0.0	0.0	2.0	24.0	0.0	0.0	79.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	40.0	0.0
2	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	2.0	0.0	5.0	1.0	0.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	41.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	
4	0.0	26.0	0.0	0.0	14.0	16.0	4.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	36.0	4.0	32.0	36.0	2.0	1.0	20.0	0.0	0.0	13.0	3.0	5.0	17.0	2.0	4.0	0.0	0.0	0.0	0.0	

5 rows x 163 columns

Substituição das quantidades do tipo de lixo para localizado=1 não localizado=0

```
1 dataT1.loc[dataT1.G1>1, 'G1']=1
2 dataT1.loc[dataT1.G3>1, 'G3']=1
3 dataT1.loc[dataT1.G4>1, 'G4']=1
4 dataT1.loc[dataT1.G5>1, 'G5']=1
5 dataT1.loc[dataT1.G7>1, 'G7']=1
6 dataT1.loc[dataT1.G8>1, 'G8']=1
7 dataT1.loc[dataT1.G9>1, 'G9']=1
8 dataT1.loc[dataT1.G10>1, 'G10']=1
9 dataT1.loc[dataT1.G11>1, 'G11']=1
10 dataT1.loc[dataT1.G12>1, 'G12']=1
11 dataT1.loc[dataT1.G13>1, 'G13']=1
12 dataT1.loc[dataT1.G14>1, 'G14']=1
13 dataT1.loc[dataT1.G15>1, 'G15']=1
14 dataT1.loc[dataT1.G16>1, 'G16']=1
15 dataT1.loc[dataT1.G17>1, 'G17']=1
16 dataT1.loc[dataT1.G18>1, 'G18']=1
17 dataT1.loc[dataT1.G19>1, 'G19']=1
18 dataT1.loc[dataT1.G21>1, 'G21']=1
19 dataT1.loc[dataT1.G22>1, 'G22']=1
20 dataT1.loc[dataT1.G23>1, 'G23']=1
21 dataT1.loc[dataT1.G24>1, 'G24']=1
22 dataT1.loc[dataT1.G25>1, 'G25']=1
23 dataT1.loc[dataT1.G26>1, 'G26']=1
24 dataT1.loc[dataT1.G27>1, 'G27']=1
25 dataT1.loc[dataT1.G28>1, 'G28']=1
26 dataT1.loc[dataT1.G29>1, 'G29']=1
27 dataT1.loc[dataT1.G30>1, 'G30']=1
28 dataT1.loc[dataT1.G31>1, 'G31']=1
29 dataT1.loc[dataT1.G32>1, 'G32']=1
30 dataT1.loc[dataT1.G33>1, 'G33']=1
31 dataT1.loc[dataT1.G34>1, 'G34']=1
32 dataT1.loc[dataT1.G35>1, 'G35']=1
33 dataT1.loc[dataT1.G36>1, 'G36']=1
34 dataT1.loc[dataT1.G37>1, 'G37']=1
35 dataT1.loc[dataT1.G40>1, 'G40']=1
36 dataT1.loc[dataT1.G41>1, 'G41']=1
37 dataT1.loc[dataT1.G42>1, 'G42']=1
38 dataT1.loc[dataT1.G43>1, 'G43']=1
39 dataT1.loc[dataT1.G44>1, 'G44']=1
40 dataT1.loc[dataT1.G45>1, 'G45']=1
41 dataT1.loc[dataT1.G46>1, 'G46']=1
42 dataT1.loc[dataT1.G47>1, 'G47']=1
43 dataT1.loc[dataT1.G49>1, 'G49']=1
44 dataT1.loc[dataT1.G50>1, 'G50']=1
45 dataT1.loc[dataT1.G52>1, 'G52']=1
46 dataT1.loc[dataT1.G53>1, 'G53']=1
47 dataT1.loc[dataT1.G54>1, 'G54']=1
48 dataT1.loc[dataT1.G56>1, 'G56']=1
49 dataT1.loc[dataT1.G57>1, 'G57']=1
50 dataT1.loc[dataT1.G58>1, 'G58']=1
51 dataT1.loc[dataT1.G59>1, 'G59']=1
52 dataT1.loc[dataT1.G60>1, 'G60']=1
53 dataT1.loc[dataT1.G62>1, 'G62']=1
54 dataT1.loc[dataT1.G63>1, 'G63']=1
55 dataT1.loc[dataT1.G64>1, 'G64']=1
56 dataT1.loc[dataT1.G65>1, 'G65']=1
57 dataT1.loc[dataT1.G66>1, 'G66']=1
58 dataT1.loc[dataT1.G67>1, 'G67']=1
59 dataT1.loc[dataT1.G68>1, 'G68']=1
60 dataT1.loc[dataT1.G69>1, 'G69']=1
61 dataT1.loc[dataT1.G70>1, 'G70']=1
62 dataT1.loc[dataT1.G71>1, 'G71']=1
63 dataT1.loc[dataT1.G72>1, 'G72']=1
64 dataT1.loc[dataT1.G73>1, 'G73']=1
65 dataT1.loc[dataT1.G76>1, 'G76']=1
66 dataT1.loc[dataT1.G77>1, 'G77']=1
67 dataT1.loc[dataT1.G79>1, 'G79']=1
68 dataT1.loc[dataT1.G80>1, 'G80']=1
69 dataT1.loc[dataT1.G82>1, 'G82']=1
70 dataT1.loc[dataT1.G83>1, 'G83']=1
71 dataT1.loc[dataT1.G84>1, 'G84']=1
72 dataT1.loc[dataT1.G85>1, 'G85']=1
73 dataT1.loc[dataT1.G86>1, 'G86']=1
74 dataT1.loc[dataT1.G87>1, 'G87']=1
75 dataT1.loc[dataT1.G88>1, 'G88']=1
76 dataT1.loc[dataT1.G89>1, 'G89']=1
77 dataT1.loc[dataT1.G90>1, 'G90']=1

```

```
1 dataT1.head()

   G1  G3  G4  G5  G7  G8  G9  G10  G11  G12  G13  G14  G15  G16  G17  G18  G19  G21  G22  G23  G24  G25  G26  G27  G28  G29  G30  G31  G32  G33  G34  G35  G36  G37  G40  G41  G42  G43  G44
0  0.0  1.0  1.0  0.0  1.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  1.0  1.0  0.0  1.0  1.0  1.0  1.0  0.0  1.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
2  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0  1.0  0.0  1.0  1.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  1.0  0.0  0.0  1.0  1.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  0.0  0.0  1.0  1.0  1.0  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0

5 rows × 163 columns
```

```
1 dataT1['CLASSE'].value_counts()

LIMPO      128
POLUIDO     125
1
Name: CLASSE, dtype: int64
```

---

```
[21] 1 dataT1['CLASSE'] = dataT1['CLASSE'].map({'LIMPO': 0, 'POLUIDO': 1})
2 dataT1['CLASSE'].value_counts()

0.0      128
1.0      125
Name: CLASSE, dtype: int64
```

```
[22] 1 X = dataT1.drop('CLASSE', axis=1)
      2 y = dataT1[['CLASSE']]
```

```
[22] 1 X = dataT1.drop('CLASSE', axis=1)
      2 y = dataT1[['CLASSE']]
```

```
[72] 1 from sklearn.model_selection import train_test_split
      2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
```

	CLASSE
221	1.0
50	1.0
197	1.0
247	1.0
156	0.0

## Modelo Sequencial

### Declarando as Camadas

```
[74] 1 model = keras.Sequential(layers.Dense(161, activation='sigmoid', input_shape=[162]))
      2 model.add(layers.Dense(1, activation='sigmoid'))
      3 model.summary()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 161)	26243
dense_7 (Dense)	(None, 1)	162
Total params: 26,405		
Trainable params: 26,405		
Non-trainable params: 0		

### Compilando o Modelo

```
[75] 1 model.compile(loss='binary_crossentropy', metrics=['binary_accuracy'],)
```

### Treinando o Modelo

```
[76] 1 history = model.fit(
      2     X_train, y_train,
      3     batch_size=32,
      4     validation_split=0.4,
      5     epochs=20,
      6     verbose=1,
      7 )
```

```
Epoch 1/20
4/4 [=====] - 1s 87ms/step - loss: 0.6415 - binary_accuracy: 0.6604 - val_loss: nan - val_binary_accuracy: 0.7606
Epoch 2/20
4/4 [=====] - 0s 13ms/step - loss: 0.5391 - binary_accuracy: 0.7358 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 3/20
4/4 [=====] - 0s 10ms/step - loss: 0.4870 - binary_accuracy: 0.8491 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 4/20
4/4 [=====] - 0s 10ms/step - loss: 0.4458 - binary_accuracy: 0.9151 - val_loss: nan - val_binary_accuracy: 0.8451
Epoch 5/20
4/4 [=====] - 0s 10ms/step - loss: 0.4163 - binary_accuracy: 0.9057 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 6/20
4/4 [=====] - 0s 9ms/step - loss: 0.3915 - binary_accuracy: 0.9434 - val_loss: nan - val_binary_accuracy: 0.8451
Epoch 7/20
4/4 [=====] - 0s 9ms/step - loss: 0.3665 - binary_accuracy: 0.9434 - val_loss: nan - val_binary_accuracy: 0.8169
Epoch 8/20
4/4 [=====] - 0s 11ms/step - loss: 0.3480 - binary_accuracy: 0.9151 - val_loss: nan - val_binary_accuracy: 0.8451
Epoch 9/20
4/4 [=====] - 0s 9ms/step - loss: 0.3285 - binary_accuracy: 0.9434 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 10/20
4/4 [=====] - 0s 9ms/step - loss: 0.3135 - binary_accuracy: 0.9528 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 11/20
4/4 [=====] - 0s 10ms/step - loss: 0.2991 - binary_accuracy: 0.9434 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 12/20
4/4 [=====] - 0s 14ms/step - loss: 0.2839 - binary_accuracy: 0.9528 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 13/20
4/4 [=====] - 0s 9ms/step - loss: 0.2714 - binary_accuracy: 0.9528 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 14/20
4/4 [=====] - 0s 12ms/step - loss: 0.2608 - binary_accuracy: 0.9528 - val_loss: nan - val_binary_accuracy: 0.8592
Epoch 15/20
4/4 [=====] - 0s 16ms/step - loss: 0.2487 - binary_accuracy: 0.9528 - val_loss: nan - val_binary_accuracy: 0.8310
Epoch 16/20
4/4 [=====] - 0s 10ms/step - loss: 0.2381 - binary_accuracy: 0.9528 - val_loss: nan - val_binary_accuracy: 0.8592
Epoch 17/20
4/4 [=====] - 0s 13ms/step - loss: 0.2281 - binary_accuracy: 0.9623 - val_loss: nan - val_binary_accuracy: 0.8592
Epoch 18/20
4/4 [=====] - 0s 15ms/step - loss: 0.2199 - binary_accuracy: 0.9434 - val_loss: nan - val_binary_accuracy: 0.8592
Epoch 19/20
4/4 [=====] - 0s 9ms/step - loss: 0.2105 - binary_accuracy: 0.9623 - val_loss: nan - val_binary_accuracy: 0.8451
Epoch 20/20
4/4 [=====] - 0s 9ms/step - loss: 0.2030 - binary_accuracy: 0.9528 - val_loss: nan - val_binary_accuracy: 0.8592
```

## Curva de Aprendizado

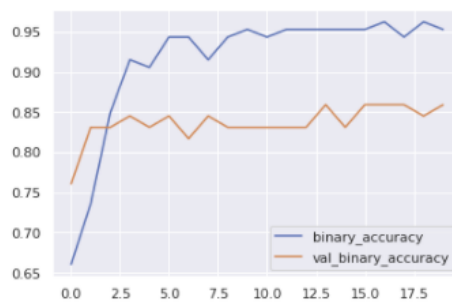
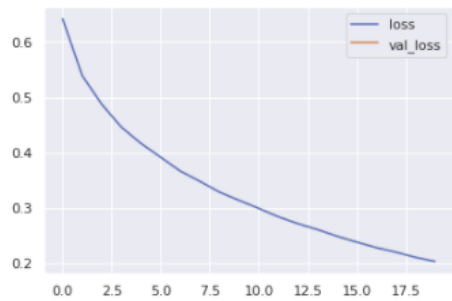
```
[77] 1 def display_hist(history):
2     history_df = pd.DataFrame(history.history)
3     display(history_df.head())
4     # Start the plot at epoch 0
5     history_df.loc[0:, ['loss', 'val_loss']].plot()
6     history_df.loc[0:, ['binary_accuracy', 'val_binary_accuracy']].plot()
7
8     print(("Best Validation Loss: {:.4f}" + \
9           "\nBest Validation Accuracy: {:.4f}" + \
10          .format(history_df['val_loss'].min(),
11                  history_df['val_binary_accuracy'].max()))
12     return
13
14
```

```
[78] 1 display_hist(history)
```

	loss	binary_accuracy	val_loss	val_binary_accuracy
0	0.641506	0.660377	NaN	0.760563
1	0.539056	0.735849	NaN	0.830986
2	0.487034	0.849057	NaN	0.830986
3	0.445777	0.915094	NaN	0.845070
4	0.416320	0.905660	NaN	0.830986

Best Validation Loss: nan

Best Validation Accuracy: 0.8592





## Resultados

Podemos então avaliar os resultados do nosso modelo fazendo a predição do conjunto de teste selecionado anteriormente.

Aqui podemos empregar as métricas usuais do `Scikit-Learn`. A predição, tendo um único neurônio de saída com a função sigmóide (ou logística) devolve um único valor entre  $[0, 1]$  e podemos entender esse valor como a chance de ser a classe 1.

```
[79] 1 from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
    2 def print_results(y_test, y_pred):
    3     print('Matriz de Confusão: \n', confusion_matrix(y_test, y_pred))
    4     print(classification_report(y_test, y_pred))
    5     print('Acuracidade: ', accuracy_score(y_test, y_pred))
    6     return
```

```
[80] 1 y_pred = model.predict(X_test) > 0.5
    2
    3 print_results(y_test, y_pred)
    4
```

```
Matriz de Confusão:
[[31 10]
 [ 1 35]]
      precision    recall  f1-score   support

    0.0         0.97     0.76     0.85         41
    1.0         0.78     0.97     0.86         36

 accuracy
macro avg         0.87     0.86     0.86         77
weighted avg         0.88     0.86     0.86         77

Acuracidade:  0.8571428571428571
```

## Conclusão

O objetivo de acuracidade de 0.8 foi atingido com o valor de 0.857, no entanto o dataset exigiu um `validation_split=0.4` ao invés de 0.3 do modelo original, por retornar NAN no Loss para valores abaixo de 0.4.