MPRTC

September 18, 2020

1 Medicina Personalizada - Redefinindo o Tratamento de Câncer

Muito tem sido dito durante os últimos anos sobre como a medicina de precisão e, mais concretamente, como o teste genético, vai provocar disrupção no tratamento de doenças como o câncer.

Mas isso ainda está acontecendo apenas parcialmente devido à enorme quantidade de trabalho manual ainda necessário. Neste projeto, tentaremos levar a medicina personalizada ao seu potencial máximo.

Uma vez sequenciado, um tumor cancerígeno pode ter milhares de mutações genéticas. O desafio é distinguir as mutações que contribuem para o crescimento do tumor das mutações.

Atualmente, esta interpretação de mutações genéticas está sendo feita manualmente. Esta é uma tarefa muito demorada, onde um patologista clínico tem que revisar manualmente e classificar cada mutação genética com base em evidências da literatura clínica baseada em texto.

Para este projeto, o MSKCC (Memorial Sloan Kettering Cancer Center) está disponibilizando uma base de conhecimento anotada por especialistas, onde pesquisadores e oncologistas de nível mundial anotaram manualmente milhares de mutações.

Neste projeto, você vai desenvolver um algoritmo de Aprendizado de Máquina que, usando essa base de conhecimento como uma linha de base, classifica automaticamente as variações genéticas.

O dataset completo pode ser encontrado em: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data

Este projeto faz parte do curso Machine Learning da Data Science Academy

Preparando as bibliotecas a serem utilizadas

[1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np

from keras.models import Sequential,Input,Model, load_model
from keras.metrics import AUC
from keras.layers import Dense
from keras.utils import plot_model,to_categorical
from keras.optimizers import SGD
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
import keras.regularizers as regularizers
```

```
from tensorflow.compat.v1.keras.backend import␣
 ↪set_session,clear_session,get_session
from tensorflow.compat.v1 import Session
import gc

from sklearn.metrics import balanced_accuracy_score, accuracy_score,␣
 ↪roc_auc_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import scikitplot as skplt
import tensorflow as tf
import re
import string
import scipy.sparse as sp
from scipy.stats import boxcox, zscore

from os import listdir

import json

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.collocations import BigramAssocMeasures, BigramCollocationFinder
from nltk import pos_tag
from nltk.tokenize import regexp_tokenize

import pickle
from collections import Counter
from functools import reduce
from wordcloud import WordCloud

from numba import cuda
```

[2]:
```
#Teste de GPU
!nvidia-smi
```

```
Mon Sep 14 21:53:00 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 452.06       Driver Version: 452.06       CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 105... WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   39C    P8    N/A /  N/A |     78MiB /  4096MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[3]:
```
#Checando a quantidade de GPU's disponíveis
print("Número Disponível de GPUs: ", len(tf.config.experimental.
 ↪list_physical_devices('GPU')))
```

```
Número Disponível de GPUs:  1
```

Both, training and test, data sets are provided via two different files. One (training/test_variants) provides the information about the genetic mutations, whereas the other (training/test_text) provides the clinical evidence (text) that our human experts used to classify the genetic mutations. Both are linked via the ID field.

Therefore the genetic mutation (row) with ID=15 in the file training_variants, was classified using the clinical evidence (text) from the row with ID=15 in the file training_text

- **training_variants** - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the aminoacid change for this mutations), Class (1-9 the class this genetic mutation has been classified on)
- **training_text** - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)
- **test_variants** - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the aminoacid change for this mutations)
- **test_text** - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)

Biblioteca de funções a ser utilizadas

[2]:
```
#função para ler os arquivos variant disponibilizados e convertendo em␣
 ↪dataframe
def convert_variant_df(read):
    lista = open(read, "r",encoding="utf8").readlines()
    #Esta lista possui \n junto ao texto, entao vamos remover
    lista_nova = [texto.split(sep="\n")[0].split(",") for texto in lista]
    df = pd.DataFrame(lista_nova[1:],columns=lista_nova[0])
    return(df)

#função para converter os demais arquivos em dataframe
```

```python
def convert__df(read):
    #separa o texto pelo delimitador ||
    lista = re.split('([0-9]+)(\|\|)',open(read, "r",encoding="utf8").read())
    #Remove da lista os elementos ||
    lista = [elemento for elemento in lista if elemento != "||"]
    #Detecta o titulo do df
    titulo = lista[0].split("\n")[0].split(",")
    lista_nova= [[lista[index+1],lista[index+2]] for index in range(0,len(lista[1:])-1,2)]

    df = pd.DataFrame(lista_nova,columns=titulo)
    return(df)

# Criando uma função que retorna um dataframe de descrição de dados (tal qual a função describe do pacote explore do R)
def explore_describe(df):
    df_out = pd.DataFrame(columns = ['variable','type','na' ,'na_pct' ,'unique','min', 'quat25','median','mean', \
                                                 'quat75','max','std','skewness','kurtosis','media_desvio'])
    df_out['variable'] = df.columns
    df_out['type'] = df.dtypes.values
    df_out['na'] = [sum(df[coluna].isna()) for coluna in df.columns]
    df_out['na_pct'] = [str(round(100*sum(df[coluna].isna())/df.shape[0],1))+'%' for coluna in df.columns]
    df_out['unique'] = [len(df[coluna].unique()) for coluna in df.columns]
    df_out['min']  = [round(min(df[coluna]),2) if 'int' in str(df[coluna].dtype) or 'float' in str(df[coluna].dtype) else '-' for coluna in df.columns]
    df_out['mean'] = [round(df[coluna].mean(),2) if 'int' in str(df[coluna].dtype) or \
                      'float' in str(df[coluna].dtype) else '-' for coluna in df.columns]
    df_out['max']  = [round(max(df[coluna]),2) if 'int' in str(df[coluna].dtype) or 'float' in str(df[coluna].dtype) else '-' for coluna in df.columns]
    df_out['std'] = [round(df[coluna].std(),2) if 'int' in str(df[coluna].dtype) or \
                      'float' in str(df[coluna].dtype) else '-' for coluna in df.columns]
    df_out['quat25'] = [round(df[coluna].quantile(0.25),2) if 'int' in str(df[coluna].dtype) or \
                      'float' in str(df[coluna].dtype) else '-' for coluna in df.columns]
    df_out['quat75'] = [round(df[coluna].quantile(0.75),2) if 'int' in str(df[coluna].dtype) or \
                      'float' in str(df[coluna].dtype) else '-' for coluna in df.columns]
```

```
    df_out['median'] = [round(df[coluna].quantile(0.5),2) if 'int' in␣
↪str(df[coluna].dtype) or \
                        'float' in str(df[coluna].dtype) else '-' for coluna in␣
↪df.columns]
    df_out['skewness'] = [round(df[coluna].skew(),2) if 'int' in str(df[coluna].
↪dtype) or \
                          'float' in str(df[coluna].dtype) else '-' for coluna␣
↪in df.columns]
    df_out['kurtosis'] = [round(df[coluna].kurt(),2) if 'int' in str(df[coluna].
↪dtype) or \
                          'float' in str(df[coluna].dtype) else '-' for coluna␣
↪in df.columns]

    df_out_media_desvio_list = []
    for coluna in df.columns:
        if(('int' in str(df[coluna].dtype)) or ('float' in str(df[coluna].
↪dtype)) ):
            if((all(df[coluna] == 0)) or (df[coluna].std() == 0)):
                df_out_media_desvio_list.append(0)
            else:
                df_out_media_desvio_list.append(round(df[coluna].mean()/
↪df[coluna].std(),2))
        else:
            df_out_media_desvio_list.append('-')

    df_out['media_desvio'] = df_out_media_desvio_list
    return(df_out)

#função para remover caracteres não ASCII
def removeNoAscii(s):
    return "".join(i for i in s if ord(i) < 128)

#Função para gerar corpus (lista de documentos)
def corpusnization(text):
    #Removendo a pontuação e tokenizando
    nopunct_token = regexp_tokenize(text.lower(),"[\w']+")

    #Removendo stopwords
    token_no_stopwords = [word for word in nopunct_token if word not in␣
↪stopwords.words('english')]

    #Stemming
    #cooking -> cook
    token_stem = [PorterStemmer().stem(token) for token in token_no_stopwords]

    #Lemmatization
```

```
    #mice -> mouse
    token_final = [WordNetLemmatizer().lemmatize(token) for token in token_stem]
    return(token_stem)

#função para criar o set de palavras (em ordem alfabética e sem repetição)
def cria_listagem_palavras(corpus):
    listagem = set()
    for i in corpus:
        listagem = listagem.union(set(i))
    return(listagem)

#Esta função irá retornar uma estrutra para o dicionario de palavras,␣
 ↪atribuindo indices a elas
def cria_dicionario_palavras(listagem):
    dicionario = {i:j for j,i in enumerate(listagem)}
    return(dicionario)

# Vamos contar a quantidade de palavras em cada elemento do corpus
def conta_palavras_corpus(elem_corpus):
    return dict(Counter(elem_corpus))

#Função para criar o dicionario de palavras
def df_dict(listagem,bag_words_corpus):
    return {token: sum([token in doc.keys() for doc in bag_words_corpus]) for␣
 ↪token in listagem}



#tf(termo,documento) = contagem de termo em documento / número de palavras em␣
 ↪documento
#idf (termo) = log (N / (df + 1))

#tf-idf (termo, documento) = tf (termo, documento) * log (N / (df + 1))
def calcula_tf_e_idf(bow, df,N):
    #Calculando a frequencia do termo para cada documento
    tf = [{key:t/sum(documento.values()) for key,t in documento.items()} for␣
 ↪documento in bow]
    idf = {chave: np.log(N/(valor+1))  for chave,valor in df.items()}
    return (tf,idf)


def tf_idf(lista_tf,dict_idf):
    return [{chave: tf*dict_idf[chave] for chave,tf in doc.items()} for doc in␣
 ↪lista_tf]

# Função para criar nuvem de palavras
def word_cloud_plot(classe,df,bag_words):
```

```
    indices = df[df["Class"] == classe].index
    bag_words_filtrada = [bag_words[i] for i in indices]
    dicionario_classe = reduce(lambda x,y: Counter(x) + Counter(y),␣
 ↪bag_words_filtrada)
    wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate_from_frequencies(dicionario_classe)
    return(wordcloud)


#Função que cria uma nuvem de palvras baseado não na frequencia, mas sim, no TF␣
 ↪IDF
def word_cloud_plot_tfidf(classe,df,lista_tfidf):
    indices = df[df["Class"] == classe].index
    lista_tfidf_filtrada = [lista_tfidf[i] for i in indices]
    dicionario_classe_tf_idf = reduce(lambda x, y: dict((chave, valor +␣
 ↪y[chave]) if chave in y.keys() else (chave, valor) for chave, valor in x.
 ↪items()), lista_tfidf_filtrada)
    wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).
 ↪generate_from_frequencies(dicionario_classe_tf_idf)
    return(wordcloud)

#Função para plotar a nuvem e palavras
def plota_wordcloud(func,df,lista_dicionario_palavras):
    fig = plt.figure(figsize=(20,10))
    for i in range(1,10):
        ax = fig.add_subplot(3,3,i)
        ax.imshow(func(i,df,lista_dicionario_palavras))
        ax.set_title(str(i))
        plt.axis("off")
    plt.show()

# Para criar o modelo base, precisamos de uma base de dados inicial. Para isso,␣
 ↪vamos criar o dataset usando as palavras da listagem
def matriz_esparca(dicio,tfidf_,N_):
    S = sp.dok_matrix((N_,len(dicio)),dtype = np.float32)
    for i,doc in enumerate(tfidf_):
        for chave,valor in doc.items():
            S[i,dicio[chave]] = valor

    return S

#Função para converter a matriz esparça em tfSparce
def convert_matriz_esparca_tfSparce(M):
    coo = M.tocoo()
```

```
    indices = np.mat([coo.row, coo.col]).transpose()
    return(tf.SparseTensor(indices, coo.data, coo.shape))

#Função para criar sempre os mesmos arquivos de treino e validação, para não␣
 ↪termos diferenças nos treinamentos
def train_test_valid_split(df_treino):
    #Dividindo os dados de treino e teste, para verificar o quao bom nosso␣
 ↪modelo está
    #O stratify servirá para melhor dividirmos os dados
    x_train, x_test, y_train, y_test = train_test_split(df_treino.drop(columns␣
 ↪= "Class"), df_treino["Class"],
                                                        test_size=0.3,␣
 ↪random_state=42,stratify = df_treino["Class"])
    x_train,x_valid,y_train,y_valid =  train_test_split(x_train, y_train,␣
 ↪test_size=0.2, random_state=42, stratify = y_train)

    return (x_train,x_test,x_valid,y_train,y_test,y_valid)

#Variáveis de entrada
def dados_modelo_treino_valid(x_train,x_test,x_valid,df_training,d,TFIDF,n):
    X = matriz_esparca(d,TFIDF,n)

    X_train = convert_matriz_esparca_tfSparce(X[x_train.index,:])
    X_test = convert_matriz_esparca_tfSparce(X[x_test.index,:])
    X_valid = convert_matriz_esparca_tfSparce(X[x_valid.index,:])

    #Classes (vamos colocar as classes de 0 a 8)
    Y = df_training.Class.apply(lambda x: x-1).to_numpy()
    Y_train = to_categorical(Y[x_train.index])
    Y_test = to_categorical(Y[x_test.index])
    Y_valid = to_categorical(Y[x_valid.index])

    return(X,X_train,X_test,X_valid,Y_train,Y_test,Y_valid)

def start_training(modelo, saving_checkpoint_path, nome_modelo, X_train,␣
 ↪Y_train, X_valid, Y_valid,
                   batch_size = 20, saved_checkpoint_path = None,␣
 ↪modelo_checkpoint = None,
                   earlyStopping = None,  reduce_lr_loss = None, initial_epoch␣
 ↪= 0):

    #Verifica se há um ModelCkeckpoint customizado
    if modelo_checkpoint is None:
        modelo_checkpoint = ModelCheckpoint("".
 ↪join((saving_checkpoint_path,nome_modelo,".hdf5")),
```

```
                                            save_best_only=True,␣
↪monitor='val_loss', mode='min')

    #Verifica se há um EarlyStopping customizado
    if earlyStopping is None:
        earlyStopping = EarlyStopping(monitor='val_loss', patience=10,
                                      verbose=1, mode='min')

    #Verifica se há um ReduceLROnPlateau customizado
    if reduce_lr_loss is None:
        reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1,␣
↪patience=7,
                                           verbose=1, min_delta=1e-4,␣
↪mode='min')

    #Verifica se o treinamento começou e já existe um checkpoint salvo
    versao_history = ""
    if saved_checkpoint_path is not None:
        # Load model:
        modelo = load_model(saved_checkpoint_path)
        # Finding the epoch index from which we are resuming
        initial_epoch = get_init_epoch(saved_checkpoint_path)
        arquivos_saved_checkpoint_path = listdir(saved_checkpoint_path)
        versao_history = [i for i in listdir(saved_checkpoint_path) if "".
↪join(("history",nome_modelo)) in i]

    modelo.fit(X_train,Y_train,epochs=150,
                    validation_data = (X_valid,Y_valid),
                    batch_size=batch_size,shuffle=True, initial_epoch=␣
↪initial_epoch,
                    callbacks=[earlyStopping, modelo_checkpoint, reduce_lr_loss])

    #Convertendo para float o lr, para que possa ser salvo
    modelo.history.history["lr"] = [float(i) for i in modelo.history.
↪history["lr"] ]
    with open("".
↪join((saving_checkpoint_path,"history_",nome_modelo,versao_history,".
↪json")), 'w') as file:
        json.dump(modelo.history.history, file)

    return modelo

def plot_matriz_confusao(modelo,xTeste,yTeste):
    skplt.metrics.plot_confusion_matrix(y_pred=tf.argmax(modelo.
↪predict(xTeste), axis = 1)+1,
                                     y_true=yTeste.argmax(axis=1)+1,
```

```
                                    figsize=(12,12))
    plt.title("".join(('Matriz de Confusão - Acuracia:',
             str(round(accuracy_score(y_pred=tf.argmax(modelo.predict(xTeste),␣
 ↪axis = 1)+1,
                                        y_true=yTeste.argmax(axis=1)+1),2)),
             ' - Acurácia balanceada:',
             str(round(balanced_accuracy_score(y_pred=tf.argmax(modelo.
 ↪predict(xTeste), axis = 1)+1,
                                        y_true=yTeste.argmax(axis=1)+1),2)))))
    plt.show()

    #Função para remover palavras indesejadas no corpus
def remove_palavras(listagem,bag_words_corpus,palavras_remover):
    l = listagem.copy()
    bow_corpus = bag_words_corpus.copy()
    palavras_remover = set(palavras_remover)
    print("Conjunto de palavras a remover concluído")
    l = [palavra for palavra in l if palavra not in palavras_remover]
    print("Listagem nova concluida")
    bow_corpus = [{chave: valor for chave,valor in doc.items() if chave not in␣
 ↪palavras_remover} for doc in bow_corpus]
    print("BOW concluído")
    d = cria_dicionario_palavras(l)
    print("Dicionário novo concluído")
    n = len(d)
    doc_freq = df_dict(l,bow_corpus)
    print("DoqFreq novo concluído")
    TF,IDF = calcula_tf_e_idf(bow_corpus, doc_freq,n)
    print("TF e IDF novos concluídos")
    TFIDF = tf_idf(TF,IDF)
    print("TFIDF novo concluído")
    return(l,bow_corpus,d,n,doc_freq,TF,IDF,TFIDF)

#Função de plotagem do historico de treinamento: evolução da perda dos dados de␣
 ↪treino e validação
def plot_treinamento(historico):
    plt.figure(figsize = (12,8))
    plt.subplot(2,1,1)
    plt.plot(historico["loss"])
    plt.plot(historico["val_loss"])

    plt.legend(("Dados de Treino","Dados de Validação"))

    plt.title("Entropia Cruzada Categórica")

    plt.subplot(2,1,2)
```

```
    plt.plot(historico["categorical_accuracy"])
    plt.plot(historico["val_categorical_accuracy"])
    plt.title("Acurácia Categórica")
    plt.legend(("Dados de Treino","Dados de Validação"))

    plt.show()

    #Função para resetar o keras
def reset_keras():
    sess = get_session()
    clear_session()
    sess.close()
    sess = get_session()

    try:
        del classifier
    except:
        pass

    print(gc.collect())


    config = tf.compat.v1.ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 1
    config.gpu_options.visible_device_list = "0"
    set_session(Session(config=config))
```

[23]:
```
training_text = "./data_files/training_text"
training_variants = "./data_files/training_variants"
```

Lendo os arquivos de entrada e convertendo os dados lidos em dataframes

[26]:
```
df_training_text = convert__df(training_text)
df_training_variants = convert_variant_df(training_variants)
```

Função para exploração inicial de dados

[28]:
```
df_training = df_training_text.merge(right=df_training_variants,on = 'ID').
 ↪drop(columns = "ID")
df_training
```

[28]:
```
                                                  Text    Gene  \
0     Cyclin-dependent kinases (CDKs) regulate a var...  FAM58A
1      Abstract Background  Non-small cell lung canc...     CBL
2      Abstract Background  Non-small cell lung canc...     CBL
3     Recent evidence has demonstrated that acquired...     CBL
4     Oncogenic mutations in the monomeric Casitas B...     CBL
...                                                 ...     ...
```

```
3316  Introduction  Myelodysplastic syndromes (MDS) ...   RUNX1
3317  Introduction  Myelodysplastic syndromes (MDS) ...   RUNX1
3318  The Runt-related transcription factor 1 gene (...   RUNX1
3319  The RUNX1/AML1 gene is the most frequent targe...   RUNX1
3320  The most frequent mutations associated with le...   RUNX1

                 Variation Class
0     Truncating Mutations     1
1                    W802*     2
2                    Q249E     2
3                    N454D     3
4                    L399V     4
...                    ...   ...
3316                 D171N     4
3317                 A122*     1
3318               Fusions     1
3319                  R80C     4
3320                  K83E     4

[3321 rows x 4 columns]
```

[30]:
```
#Checando os dados iniciais
explore_describe(df_training)
```

[30]:
```
    variable    type  na na_pct  unique min quat25 median mean quat75 max std  \
0       Text  object   0   0.0%    1921   -      -      -    -      -   -   -
1       Gene  object   0   0.0%     264   -      -      -    -      -   -   -
2  Variation  object   0   0.0%    2996   -      -      -    -      -   -   -
3      Class  object   0   0.0%       9   -      -      -    -      -   -   -

  skewness kurtosis media_desvio
0        -        -            -
1        -        -            -
2        -        -            -
3        -        -            -
```

Vamos salvar os df no formato csv para facilitar o carregamento posterior

[ ]:
```
df_training.to_csv("df_training.csv")
```

Vamos importar os arquivos csv

[2]:
```
df_training = pd.read_csv("df_training.csv")
```

[32]:
```
# Primeiramente, vamos criar a frequencia de cada termo
textos_treino = df_training.Text
```

[34]:
```
textos_treino_limpa = textos_treino.map(lambda x: removeNoAscii(x))
```

[36]:
```
textos_treino_limpa.head()
```

[36]:
```
0    Cyclin-dependent kinases (CDKs) regulate a var...
1     Abstract Background  Non-small cell lung canc...
2     Abstract Background  Non-small cell lung canc...
3    Recent evidence has demonstrated that acquired...
4    Oncogenic mutations in the monomeric Casitas B...
Name: Text, dtype: object
```

Criando gerando o CORPUS para tratamento dos dados

[8]:
```
#Removendo a pontuação, stopwords, realizando stemming e lemmatization (essa␣
 ↪célula pode demorar algumas horas para ser executada)
treino_corpus = [corpusnization(texto) for texto in textos_treino_limpa]
```

[9]:
```
#Armazenando os corpus (como a célula acima demora algumas horas, será␣
 ↪necessário salvar os dados)
with open('treino_corpus.pkl', 'wb') as f:
        pickle.dump(treino_corpus, f)
```

1.1 Carregando os dataframe e os corpus

[3]:
```
#Carregando novamente os dados. Por demorar para realizar a função␣
 ↪corpusnization para tanto os dados de treino quanto os dados de teste,
#criamos um checkpoint aqui para dar continuidade ao trabalho mais rapidamente

with open('treino_corpus.pkl', 'rb') as f:
        treino_corpus = pickle.load(f)

df_training = pd.read_csv("df_training.csv")
```

[7]:
```
treino_corpus[1]
```

[7]:
```
['abstract',
 'background',
 'non',
 'small',
 'cell',
 'lung',
 'cancer',
 'nsclc',
 'heterogen',
 'group',
 'disord',
 'number',
 'genet',
 'proteom',
 'alter',
 'c',
 'cbl',
```

```
'e3',
'ubiquitin',
'ligas',
'adaptor',
'molecul',
'import',
'normal',
'homeostasi',
'cancer',
'determin',
'genet',
'variat',
'c',
'cbl',
'relationship',
'receptor',
'tyrosin',
'kinas',
'egfr',
'met',
'function',
'nsclc',
'method',
'find',
'use',
'archiv',
'formalin',
'fix',
'paraffin',
'embed',
'ffpe',
'extract',
'genom',
'dna',
'show',
'c',
'cbl',
'mutat',
'occur',
'somat',
'fashion',
'lung',
'cancer',
'c',
'cbl',
'mutat',
'mutual',
```

```
'exclus',
'met',
'egfr',
'mutat',
'howev',
'independ',
'p53',
'kra',
'mutat',
'normal',
'tumor',
'pairwis',
'analysi',
'signific',
'loss',
'heterozygos',
'loh',
'c',
'cbl',
'locu',
'22',
'n',
'8',
'37',
'none',
'sampl',
'reveal',
'mutat',
'remain',
'copi',
'c',
'cbl',
'c',
'cbl',
'loh',
'also',
'posit',
'correl',
'egfr',
'met',
'mutat',
'observ',
'sampl',
'use',
'select',
'c',
'cbl',
```

```
'somat',
'mutat',
's80n',
'h94i',
'q249e',
'w802',
'obtain',
'caucasian',
'taiwanes',
'african',
'american',
'sampl',
'respect',
'transfect',
'nsclc',
'cell',
'line',
'increas',
'cell',
'viabil',
'cell',
'motil',
'conclus',
'take',
'overal',
'mutat',
'rate',
'c',
'cbl',
'combin',
'somat',
'missens',
'mutat',
'loh',
'clear',
'c',
'cbl',
'highli',
'mutat',
'lung',
'cancer',
'may',
'play',
'essenti',
'role',
'lung',
'tumorigenesi',
```

```
'metastasi',
'go',
'introduct',
'us',
'alon',
'year',
'approxim',
'219',
'400',
'peopl',
'diagnos',
'lung',
'cancer',
'145',
'000',
'succumb',
'diseas',
'1',
'number',
'roughli',
'equival',
'combin',
'mortal',
'rate',
'cancer',
'breast',
'prostat',
'colon',
'liver',
'kidney',
'melanoma',
'1',
'addit',
'prognosi',
'usual',
'poor',
'five',
'year',
'surviv',
'rate',
'less',
'15',
'also',
'signific',
'ethnic',
'differ',
'lung',
```

```
'cancer',
'outcom',
'wors',
'black',
'compar',
'white',
'gender',
'differ',
'also',
'strike',
'women',
'significantli',
'better',
'prognosi',
'compar',
'men',
'number',
'genet',
'alter',
'occur',
'lung',
'cancer',
'exampl',
'nsclc',
'mutat',
'kra',
'p53',
'egfr',
'met',
'identifi',
'mani',
'pathway',
'especi',
'receptor',
'tyrosin',
'kinas',
'rtk',
'control',
'c',
'cbl',
'cbl',
'casita',
'b',
'lineag',
'lymphoma',
'mammalian',
'gene',
```

```
'locat',
'human',
'chromosom',
'11q23',
'3',
'2',
'involv',
'cell',
'signal',
'protein',
'ubiquitin',
'3',
'cbl',
'protein',
'belong',
'ring',
'finger',
'class',
'ubiquitin',
'ligas',
'e3',
'three',
'homologu',
'c',
'cbl',
'cbl',
'b',
'cbl',
'3',
'4',
'c',
'cbl',
'cbl',
'b',
'gene',
'ubiquit',
'express',
'highest',
'level',
'hematopoiet',
'tissu',
'5',
'c',
'cbl',
'consist',
'four',
'region',
```

```
'encod',
'function',
'distinct',
'protein',
'domain',
'n',
'termin',
'tyrosin',
'kinas',
'bind',
'tkb',
'domain',
'linker',
'region',
'catalyt',
'ring',
'finger',
'domain',
'prolin',
'rich',
'region',
'c',
'termin',
'ubiquitin',
'associ',
'uba',
'domain',
'also',
'overlap',
'leucin',
'zipper',
'lz',
'domain',
'3',
'tkb',
'ring',
'finger',
'domain',
'essenti',
'ligand',
'induc',
'ubiquitin',
'rtk',
'6',
'7',
'8',
'9',
```

```
'ring',
'finger',
'domain',
'requir',
'recruit',
'e2',
'ubiquitin',
'conjug',
'enzym',
'tkb',
'domain',
'includ',
'four',
'helix',
'bundl',
'4h',
'calcium',
'bide',
'ef',
'hand',
'modifi',
'sh2',
'domain',
'bind',
'phosphotyrosin',
'residu',
'3',
'10',
'11',
'12',
'addit',
'prolin',
'rich',
'region',
'c',
'cbl',
'associ',
'sh3',
'domain',
'grb2',
'indirectli',
'recruit',
'c',
'cbl',
'rtk',
'via',
'grb2',
```

```
'adaptor',
'protein',
'7',
'13',
'14',
'c',
'cbl',
'also',
'bind',
'egfr',
'act',
'e3',
'target',
'egfr',
'ubiquitin',
'degrad',
'furthermor',
'cbl',
'desensit',
'egf',
'signal',
'oppos',
'cellular',
'prolifer',
'induc',
'egf',
'15',
'egf',
'activ',
'also',
'appear',
'activ',
'tyrosin',
'kinas',
'src',
'phosphoryl',
'c',
'cbl',
'turn',
'activ',
'ubiquitin',
'degrad',
'egfr',
'16',
'17',
'18',
'recent',
```

```
'studi',
'show',
'defect',
'endocytosi',
'egfr',
'character',
'delet',
'mutant',
'point',
'mutat',
'l858r',
'wherebi',
'associ',
'c',
'cbl',
'subsequ',
'ubiquitin',
'impair',
'19',
'recent',
'first',
'human',
'c',
'cbl',
'mutat',
'report',
'acut',
'myeloid',
'leukemia',
'aml',
'patient',
'20',
'mutat',
'r420q',
'inhibit',
'fm',
'like',
'tyrosin',
'kinas',
'3',
'flt3',
'intern',
'ubiquitin',
'20',
'e3',
'activ',
'import',
```

```
'oncogenesi',
'c',
'cbl',
'dual',
'separ',
'function',
'signal',
'transduct',
'molecul',
'previous',
'shown',
'c',
'cbl',
'import',
'bind',
'crkl',
'bcr',
'abl',
'hematopoiet',
'cell',
'also',
'bind',
'modul',
'function',
'cytoskeleton',
'bind',
'protein',
'like',
'talin',
'paxillin',
'tkb',
'domain',
'import',
'bind',
'number',
'molecul',
'function',
'signal',
'transduct',
'given',
'critic',
'role',
'cbl',
'normal',
'homeostasi',
'cancer',
'hypothes',
```

```
'might',
'mutat',
'lung',
'cancer',
'studi',
'report',
'novel',
'c',
'cbl',
'somat',
'mutat',
's80n',
'h94i',
'q249e',
'w802',
'caucasian',
'taiwanes',
'african',
'american',
'lung',
'cancer',
'patient',
'respect',
'express',
'mutat',
'nsclc',
'cell',
'line',
'lead',
'increas',
'prolifer',
'cell',
'motil',
'show',
'c',
'cbl',
'mutat',
'occur',
'without',
'met',
'egfr',
'mutat',
'mutual',
'exclus',
'loh',
'c',
'cbl',
```

```
'locu',
'addit',
'c',
'cbl',
'loh',
'associ',
'either',
'met',
'egfr',
'mutat',
'thu',
'hypothes',
'c',
'cbl',
'mutat',
'might',
'contribut',
'oncogen',
'potenti',
'met',
'egfr',
'lung',
'cancer',
'go',
'method',
'ethic',
'statement',
'written',
'consent',
'research',
'human',
'subject',
'obtain',
'institut',
'review',
'board',
'univers',
'chicago',
'cover',
'research',
'perform',
'laboratori',
'follow',
'contact',
'inform',
'institut',
'review',
```

```
'board',
'univers',
'chicago',
'mcgiffert',
'hall',
'5751',
'woodlawn',
'ave',
'2nd',
'floor',
'chicago',
'il',
'60637',
'written',
'inform',
'consent',
'receiv',
'patient',
'whose',
'tissu',
'sampl',
'use',
'studi',
'tissu',
'sampl',
'lung',
'cancer',
'tissu',
'pair',
'adjac',
'normal',
'lung',
'tissu',
'obtain',
'50',
'caucasian',
'29',
'african',
'american',
'40',
'taiwanes',
'nsclc',
'patient',
'recruit',
'univers',
'chicago',
'hospit',
```

```
'chicago',
'usa',
'caucasian',
'african',
'american',
'patient',
'taipei',
'veteran',
'gener',
'hospit',
'taiwan',
'taiwanes',
'patient',
'obtain',
'appropri',
'institut',
'review',
'board',
'permiss',
'inform',
'consent',
'patient',
'119',
'sampl',
'77',
'men',
'38',
'women',
'4',
'unknown',
'age',
'diagnosi',
'rang',
'47',
'90',
'year',
'term',
'tumor',
'type',
'53',
'adenocarcinoma',
'32',
'squamou',
'cell',
'carcinoma',
'34',
'larg',
```

```
'cell',
'carcinoma',
'49',
'stage',
'14',
'stage',
'ii',
'34',
'stage',
'iii',
'13',
'stage',
'iv',
'tabl',
's1',
'cell',
'cultur',
'human',
'non',
'small',
'cell',
'lung',
'carcinoma',
'cell',
'a549',
'h358',
'maintain',
'dmem',
'rpmi',
'1640',
'respect',
'human',
'embryon',
'kidney',
'293t',
'cell',
'cultur',
'dmem',
'media',
'supplement',
'10',
'fetal',
'bovin',
'serum',
'100',
'unit',
'ml',
```

```
'penicillin',
'100',
'g',
'ml',
'streptomycin',
'invitrogen',
'carlsbad',
'ca',
'cell',
'cultur',
'37c',
'humidifi',
'incub',
'contain',
'5',
'co2',
'c',
'cbl',
'gene',
'mutat',
'analysi',
'exon',
'2',
'16',
'c',
'cbl',
'gene',
'individu',
'amplifi',
'polymeras',
'chain',
'reaction',
'pcr',
'primer',
'list',
'tabl',
's2',
'pcr',
'condit',
'1',
'cycl',
'95c',
'5',
'minut',
'35',
'cycl',
'94c',
```

```
'30',
'second',
'58c',
'30',
'second',
'72c',
'2',
'minut',
'one',
'cycl',
'72c',
'10',
'minut',
'pcr',
'product',
'treat',
'exosap',
'usb',
'corpor',
'cleveland',
'oh',
'sequenc',
'big',
'dye',
'termin',
'chemistri',
'appli',
'biosystem',
'foster',
'citi',
'ca',
'sequenc',
'perform',
'forward',
'code',
'strand',
'confirm',
'c',
'cbl',
'alter',
'perform',
'sequenc',
'revers',
'strand',
'well',
'chromatogram',
'analyz',
```

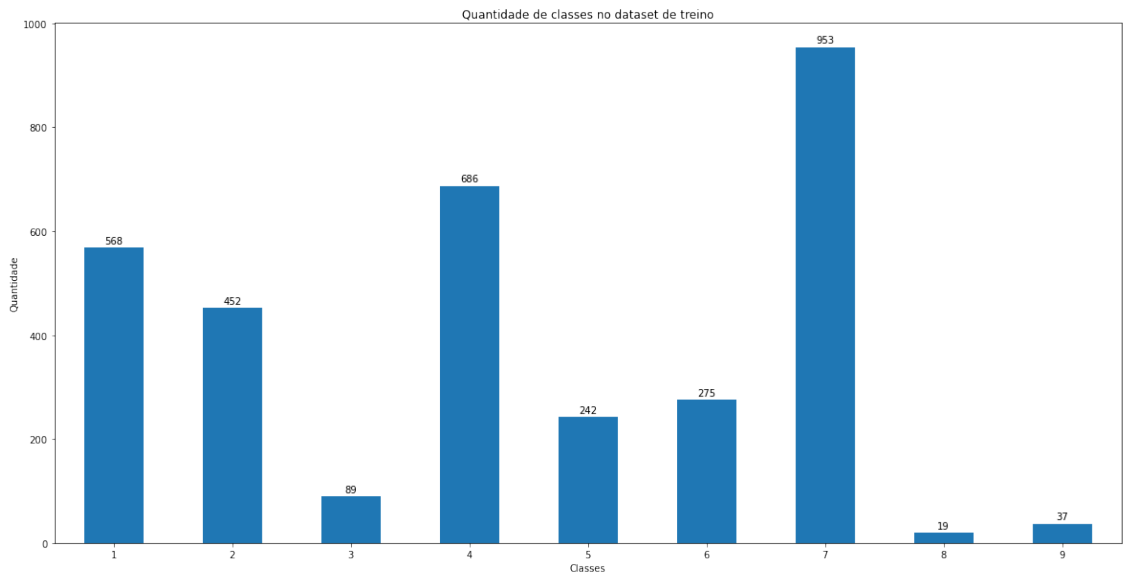
```
'mutat',
'use',
'mutat',
'surveyor',
'v2',
'61',
'softgenet',
'state',
'colleg',
'pa',
'plasmid',
'construct',
'site',
'direct',
'mutagenesi',
'wild',
'type',
'c',
'cbl',
'cdna',
'insert',
'subclon',
'paltermax',
'express',
'vector',
'use',
'xhoi',
'sali',
'restrict',
'enzym',
'site',
'promega',
'madison',
'wi',
'use',
'parent',
'plasmid',
'paltermax',
'c',
'cbl',
'tkb',
'domain',
'doubl',
'mutat',
's80n',
'h94i',
'point',
```

```
'mutat',
'q249e',
'c',
'termin',
'point',
'mutat',
'w802',
'c',
'cbl',
'creat',
'use',
'follow',
'primer',
'5',
'gctggcgctaaagaataacccaccttatatcttagac',
'3',
'5',
'ctaccagatacctaccagtatctccgtactatcttgtc',
'3',
'doubl',
'mutat',
's80n',
'h94i',
'5',
'ctttacccgactctttgagccctggtcctctttgc',
'3',
'q249e',
'5',
'cagctcctcctttggctgattgtctctggatggtgatc',
'3',
'w802',
'along',
'complementari',
'primer',
'use',
'quickchang',
'site',
'direct',
'mutagenesi',
'xl',
'kit',
'stratagen',
'la',
'jolla',
'ca',
'accord',
"manufacturer'",
```

```
 'instruct',
 'construct',
 'confirm',
 'point',
 'mutat',
 'standard',
 'dna',
 'sequenc',
 'strand',
 'loss',
 'heterozygos',
 'loh',
 'analysi',
 'five',
 'microsatellit',
 'chromosom',
 '11',
 '3',
 '11q',
 'within',
 '200',
 'kb',
 'downstream',
 'c',
 'cbl',
 'gene',
 '2',
 'control',
 'marker',
 '11p',
 'select',
 'analysi',
 'tabl',
 's3',
 'establish',
 'microsatellit',
 'marker',
 'respect',
 'primer',
 'sequenc',
 'select',
 'geneloc',
 'databas',
 ...]
```

Verificando a quantidade de classes no dataset Nota-se que a quantidade de classes está desbalanceada

[3]:
```
plt.figure(figsize=(20,10))
ax = df_training["Class"].value_counts().sort_index().plot(kind = "bar")
plt.xticks(rotation = 0)
plt.xlabel("Classes")
plt.ylabel("Quantidade")
plt.title("Quantidade de classes no dataset de treino")
rects = ax.patches

for rect, label in zip(rects, list(df_training["Class"].value_counts().
 ↪sort_index())):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width() / 2, height + 5, label,
            ha='center', va='bottom')

plt.show()
```



1.2 Criando dicionário de palavras

[5]:
```
#Usando as palavras dos dados de teste, para que não haja problemas na hora dos␣
 ↪testes
listagem = cria_listagem_palavras(treino_corpus+teste_corpus)
```

[23]:
```
pickle.dump(listagem,open("listagem.pkl","wb"))
```

[37]:
```
dicionario = cria_dicionario_palavras(listagem)
```

[39]:
```
pickle.dump(dicionario,open("dicionario.pkl","wb"))
```

1.3 Criação da Bag of Words

[9]:
```
bag_words_corpus = [conta_palavras_corpus(i) for i in treino_corpus]
N = len(bag_words_corpus)
```

[30]:
```
pickle.dump(bag_words_corpus,open("bag_of_words.plk","wb"))
```

1.4 Definindo a frequencia do documento para cada palavra

[11]:
```
DocFreq = df_dict(listagem,bag_words_corpus)
```

[25]:
```
pickle.dump(DocFreq,open("DocFreq.plk","wb"))
```

1.5 TF-IDF

Vamos criar o tf-idf para a base de dados de treino, para que possamos criar nosso dataset para passar ao modelo

[13]:
```
tf_treino,idf_treino = calcula_tf_e_idf(bag_words_corpus, DocFreq,N)
```

[26]:
```
pickle.dump(tf_treino,open("tf_treino.plk","wb"))
pickle.dump(idf_treino,open("idf_treino.plk","wb"))
```

[15]:
```
tfidf = tf_idf(tf_treino,idf_treino)
```

[27]:
```
pickle.dump(tfidf,open("tfidf.plk","wb"))
```

Criando outro checkpoint e abrindo os arquivos trabalhados

[2]:
```
tfidf = pickle.load(open("tfidf.plk","rb"))
tf_treino = pickle.load(open("tf_treino.plk","rb"))
idf_treino = pickle.load(open("idf_treino.plk","rb"))
DocFreq = pickle.load(open("DocFreq.plk","rb"))
bag_words_corpus = pickle.load(open("bag_of_words.plk","rb"))
listagem = pickle.load(open("listagem.pkl","rb"))
dicionario = pickle.load(open("dicionario.pkl","rb"))

df_training = pd.read_csv("df_training.csv")

N = len(bag_words_corpus)
```

Existe um alto volume de palavras com pouquissimas repetições

[25]:
```
#plt.figure(figsize = (15,8))
#pd.Series({i: len([key for key,valor in qtd_palavras_corpus.items() if valor␣
↪<= i]) for i in range(0,20)}).plot(kind = "line")
#plt.title("Volume de repetições de palavras no documento de treino")
#plt.xlabel("Volume de repetições")
#plt.ylabel("Quantidade de palavras")
#plt.show()
```

1.6 Word Cloud

[20]:
```
plota_wordcloud(word_cloud_plot,df_training,bag_words_corpus)
```

As palavras: * mutat; * cell; * active * números isolados;
parecem estar em todos as classes

[194]:
```
#Vamos ver se há diferença na nuvem de palavras usando o conceito de TF-IDF
plota_wordcloud(word_cloud_plot_tfidf,df_training,tfidf)
```

Quando usamos tf-idf, as palavras que aparecem tm toda são: * fig: referencia a uma figura; * et: referencia bibliografica; * al: referencia bibliografica;

1.7 Vamos criar um modelo base

[4]:
```
#Dividindo os dados de treino e teste, para verificar o quao bom nosso modelo está
x_train,x_test,x_valid,y_train,y_test,y_valid = train_test_valid_split(df_training)
```

[7]:
```
X,X_train,X_test,X_valid,Y_train,Y_test,Y_valid = dados_modelo_treino_valid(x_train,x_test,x_valid,df_training,dicionario,tfidf,N)
```

2 Vamos verificar, se os dados Gene e Variation encontram-se nos respectivos textos

[25]:
```
vetor_gene = []
vetor_variation = []
for (i,[gene,variation]) in enumerate(zip(df_training["Gene"],df_training["Variation"])):
    try:
        if(X[i,dicionario[gene.lower()]]>0):
            vetor_gene.append(True)
        else:
            vetor_gene.append(False)
```

```
    except:
        vetor_gene.append(False)

    try:
        if(X[i,dicionario[variation.lower()]]>0):
            vetor_variation.append(True)
        else:
            vetor_variation.append(False)
    except:
        vetor_variation.append(False)
```

Alguns documentos não possuem a informação de gene e variation, conforme está no dataset

[11]:
```
pd.Series(vetor_variation).value_counts()
```

[11]:
```
True     1817
False    1504
dtype: int64
```

[12]:
```
pd.Series(vetor_gene).value_counts()
```

[12]:
```
True     2985
False     336
dtype: int64
```

2.1 Criando a rede neural para o treinamento - Modelo 1

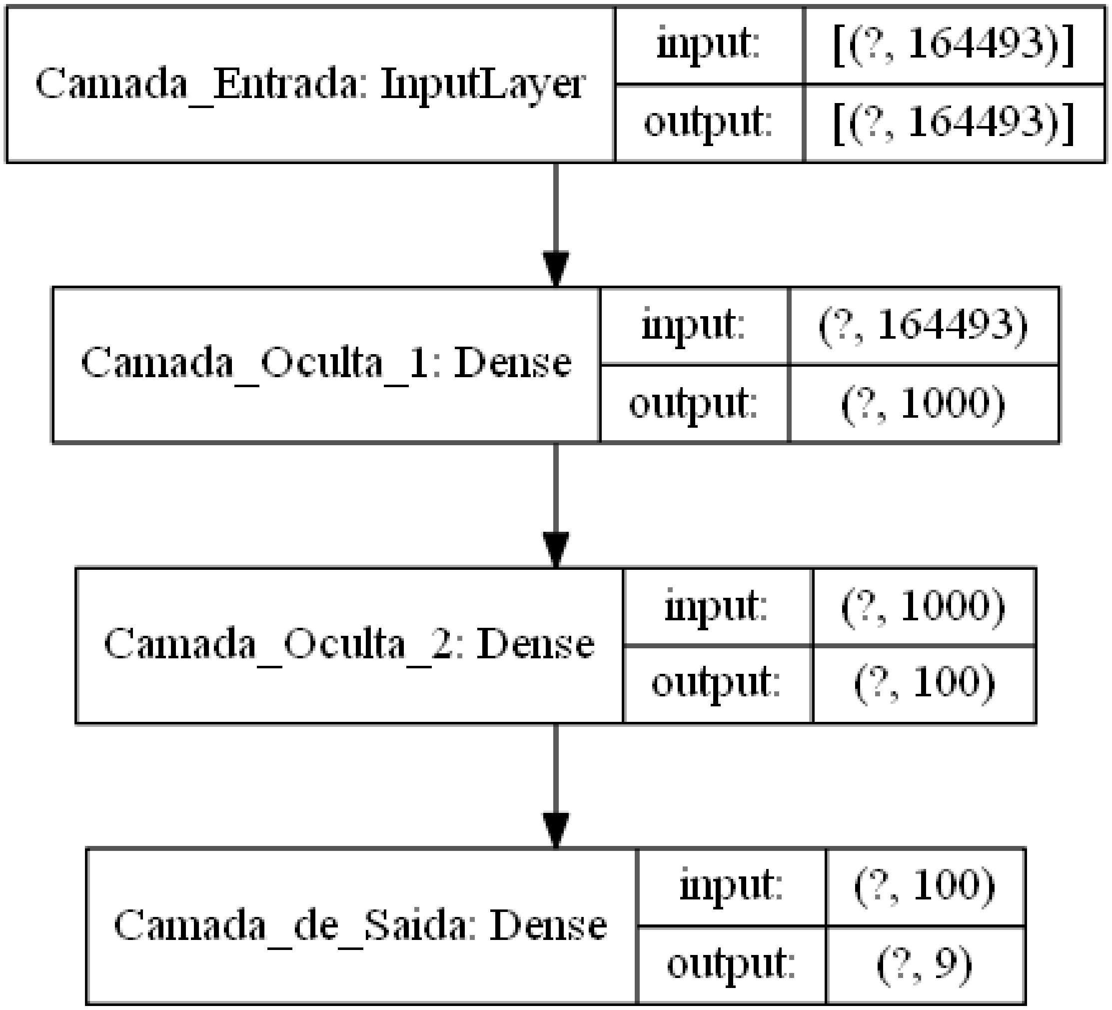
[13]:
```
X_train.shape
```

[13]:
```
TensorShape([1859, 164493])
```

[ ]:
```
#Criando o modelo para receber os dados de entrada
camada_entrada = Input(shape = (X.shape[1],), sparse=True,name =␣
 ↪"Camada_Entrada")
primeira_camada_oculta = Dense(1000,activation = 'relu',kernel_initializer =␣
 ↪'uniform',name = "Camada_Oculta_1")(camada_entrada)
segunda_camada_oculta = Dense(100,activation = 'relu',kernel_initializer =␣
 ↪'uniform',name = "Camada_Oculta_2")(primeira_camada_oculta)
camada_saida = Dense(9,activation = 'softmax',kernel_initializer =␣
 ↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo1 = Model(inputs = [camada_entrada], outputs = [camada_saida])
#modelo1.add(Dense(100,activation = 'relu',kernel_initializer = 'uniform'))
#modelo1.add(Dense(9,activation = 'softmax',kernel_initializer = 'uniform'))
```

[9]:
```
#Visualizando o primeiro modelo
plot_model(modelo1,show_shapes=True)
```

[9]:



[10]:
```
modelo1.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])
```

[11]:
```
modelo1 =␣
 ↪start_training(X_train=X_train,X_valid=X_valid,Y_train=Y_train,Y_valid=Y_valid,
               saving_checkpoint_path="./modelos/", nome_modelo="modelo1",␣
 ↪modelo= modelo1)
```

```
Epoch 1/150
93/93 [==============================] - 19s 204ms/step - loss: 1.8990 -
categorical_accuracy: 0.2781 - auc: 0.7420 - val_loss: 1.8410 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7575
Epoch 2/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8390 -
categorical_accuracy: 0.2813 - auc: 0.7553 - val_loss: 1.8415 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7578
```

```
Epoch 3/150
93/93 [==============================] - 19s 205ms/step - loss: 1.8398 -
categorical_accuracy: 0.2873 - auc: 0.7551 - val_loss: 1.8367 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7550
Epoch 4/150
93/93 [==============================] - 19s 203ms/step - loss: 1.8359 -
categorical_accuracy: 0.2824 - auc: 0.7572 - val_loss: 1.8336 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7579
Epoch 5/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8361 -
categorical_accuracy: 0.2873 - auc: 0.7564 - val_loss: 1.8373 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7569
Epoch 6/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8369 -
categorical_accuracy: 0.2873 - auc: 0.7557 - val_loss: 1.8344 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7604
Epoch 7/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8357 -
categorical_accuracy: 0.2873 - auc: 0.7568 - val_loss: 1.8369 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7587
Epoch 8/150
93/93 [==============================] - 19s 204ms/step - loss: 1.8364 -
categorical_accuracy: 0.2873 - auc: 0.7551 - val_loss: 1.8325 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7612
Epoch 9/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8355 -
categorical_accuracy: 0.2786 - auc: 0.7565 - val_loss: 1.8374 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7702
Epoch 10/150
93/93 [==============================] - 19s 205ms/step - loss: 1.8338 -
categorical_accuracy: 0.2873 - auc: 0.7580 - val_loss: 1.8292 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7619
Epoch 11/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8295 -
categorical_accuracy: 0.2873 - auc: 0.7596 - val_loss: 1.8316 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7685
Epoch 12/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8283 -
categorical_accuracy: 0.2910 - auc: 0.7604 - val_loss: 1.8303 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7746
Epoch 13/150
93/93 [==============================] - 19s 206ms/step - loss: 1.8255 -
categorical_accuracy: 0.2813 - auc: 0.7623 - val_loss: 1.8234 -
val_categorical_accuracy: 0.3484 - val_auc: 0.7785
Epoch 14/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8138 -
categorical_accuracy: 0.3093 - auc: 0.7672 - val_loss: 1.8531 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7760
```

```
Epoch 15/150
93/93 [==============================] - 19s 204ms/step - loss: 1.8042 -
categorical_accuracy: 0.3158 - auc: 0.7700 - val_loss: 1.7788 -
val_categorical_accuracy: 0.3183 - val_auc: 0.7880
Epoch 16/150
93/93 [==============================] - 19s 202ms/step - loss: 1.7667 -
categorical_accuracy: 0.3362 - auc: 0.7835 - val_loss: 1.7257 -
val_categorical_accuracy: 0.3527 - val_auc: 0.8042
Epoch 17/150
93/93 [==============================] - 6s 66ms/step - loss: 1.7373 -
categorical_accuracy: 0.3604 - auc: 0.7933 - val_loss: 1.8446 -
val_categorical_accuracy: 0.2172 - val_auc: 0.7791
Epoch 18/150
93/93 [==============================] - 19s 203ms/step - loss: 1.7147 -
categorical_accuracy: 0.3674 - auc: 0.8002 - val_loss: 1.6684 -
val_categorical_accuracy: 0.3527 - val_auc: 0.8199
Epoch 19/150
93/93 [==============================] - 19s 205ms/step - loss: 1.6800 -
categorical_accuracy: 0.3706 - auc: 0.8099 - val_loss: 1.6412 -
val_categorical_accuracy: 0.4581 - val_auc: 0.8382
Epoch 20/150
93/93 [==============================] - 19s 206ms/step - loss: 1.6476 -
categorical_accuracy: 0.3932 - auc: 0.8207 - val_loss: 1.5407 -
val_categorical_accuracy: 0.4624 - val_auc: 0.8638
Epoch 21/150
93/93 [==============================] - 6s 66ms/step - loss: 1.6341 -
categorical_accuracy: 0.3905 - auc: 0.8232 - val_loss: 1.5704 -
val_categorical_accuracy: 0.5032 - val_auc: 0.8600
Epoch 22/150
93/93 [==============================] - 6s 66ms/step - loss: 1.6037 -
categorical_accuracy: 0.4190 - auc: 0.8311 - val_loss: 1.5497 -
val_categorical_accuracy: 0.3935 - val_auc: 0.8427
Epoch 23/150
93/93 [==============================] - 6s 66ms/step - loss: 1.5598 -
categorical_accuracy: 0.4190 - auc: 0.8416 - val_loss: 1.5577 -
val_categorical_accuracy: 0.4172 - val_auc: 0.8430
Epoch 24/150
93/93 [==============================] - 19s 203ms/step - loss: 1.5683 -
categorical_accuracy: 0.4239 - auc: 0.8389 - val_loss: 1.4982 -
val_categorical_accuracy: 0.4344 - val_auc: 0.8535
Epoch 25/150
93/93 [==============================] - 19s 204ms/step - loss: 1.5540 -
categorical_accuracy: 0.4185 - auc: 0.8428 - val_loss: 1.4134 -
val_categorical_accuracy: 0.4839 - val_auc: 0.8742
Epoch 26/150
93/93 [==============================] - 6s 66ms/step - loss: 1.5158 -
categorical_accuracy: 0.4427 - auc: 0.8513 - val_loss: 1.4226 -
val_categorical_accuracy: 0.5075 - val_auc: 0.8745
```

```
Epoch 27/150
93/93 [==============================] - 19s 203ms/step - loss: 1.5230 -
categorical_accuracy: 0.4239 - auc: 0.8494 - val_loss: 1.4003 -
val_categorical_accuracy: 0.5054 - val_auc: 0.8728
Epoch 28/150
93/93 [==============================] - 6s 66ms/step - loss: 1.4613 -
categorical_accuracy: 0.4809 - auc: 0.8621 - val_loss: 1.7900 -
val_categorical_accuracy: 0.3591 - val_auc: 0.7857
Epoch 29/150
93/93 [==============================] - 19s 206ms/step - loss: 1.4615 -
categorical_accuracy: 0.4739 - auc: 0.8615 - val_loss: 1.3927 -
val_categorical_accuracy: 0.4968 - val_auc: 0.8731
Epoch 30/150
93/93 [==============================] - 19s 202ms/step - loss: 1.4042 -
categorical_accuracy: 0.4825 - auc: 0.8730 - val_loss: 1.3213 -
val_categorical_accuracy: 0.5419 - val_auc: 0.8914
Epoch 31/150
93/93 [==============================] - 6s 66ms/step - loss: 1.4870 -
categorical_accuracy: 0.4755 - auc: 0.8571 - val_loss: 1.4764 -
val_categorical_accuracy: 0.4925 - val_auc: 0.8562
Epoch 32/150
93/93 [==============================] - 6s 66ms/step - loss: 1.4710 -
categorical_accuracy: 0.4734 - auc: 0.8600 - val_loss: 1.4420 -
val_categorical_accuracy: 0.4903 - val_auc: 0.8698
Epoch 33/150
93/93 [==============================] - 6s 66ms/step - loss: 1.4312 -
categorical_accuracy: 0.4809 - auc: 0.8679 - val_loss: 1.3291 -
val_categorical_accuracy: 0.5505 - val_auc: 0.8931
Epoch 34/150
93/93 [==============================] - 6s 66ms/step - loss: 1.4038 -
categorical_accuracy: 0.4857 - auc: 0.8736 - val_loss: 1.6888 -
val_categorical_accuracy: 0.4000 - val_auc: 0.8080
Epoch 35/150
93/93 [==============================] - 6s 67ms/step - loss: 1.4469 -
categorical_accuracy: 0.4755 - auc: 0.8652 - val_loss: 1.4547 -
val_categorical_accuracy: 0.4774 - val_auc: 0.8694
Epoch 36/150
93/93 [==============================] - 6s 66ms/step - loss: 1.3684 -
categorical_accuracy: 0.4981 - auc: 0.8798 - val_loss: 1.8646 -
val_categorical_accuracy: 0.3290 - val_auc: 0.7928
Epoch 37/150
93/93 [==============================] - ETA: 0s - loss: 1.3431 -
categorical_accuracy: 0.5083 - auc: 0.8841
Epoch 00037: ReduceLROnPlateau reducing learning rate to 0.005000000074505806.
93/93 [==============================] - 6s 67ms/step - loss: 1.3431 -
categorical_accuracy: 0.5083 - auc: 0.8841 - val_loss: 1.5691 -
val_categorical_accuracy: 0.4602 - val_auc: 0.8503
Epoch 38/150
```

```
93/93 [==============================] - 19s 207ms/step - loss: 1.2069 -
categorical_accuracy: 0.5680 - auc: 0.9080 - val_loss: 1.2624 -
val_categorical_accuracy: 0.5548 - val_auc: 0.9003
Epoch 39/150
93/93 [==============================] - 6s 66ms/step - loss: 1.1428 -
categorical_accuracy: 0.5869 - auc: 0.9178 - val_loss: 1.2959 -
val_categorical_accuracy: 0.5269 - val_auc: 0.8913
Epoch 40/150
93/93 [==============================] - 19s 204ms/step - loss: 1.1316 -
categorical_accuracy: 0.5971 - auc: 0.9191 - val_loss: 1.2616 -
val_categorical_accuracy: 0.5419 - val_auc: 0.8975
Epoch 41/150
93/93 [==============================] - 6s 66ms/step - loss: 1.1265 -
categorical_accuracy: 0.5966 - auc: 0.9198 - val_loss: 1.2839 -
val_categorical_accuracy: 0.5462 - val_auc: 0.8947
Epoch 42/150
93/93 [==============================] - 19s 205ms/step - loss: 1.1206 -
categorical_accuracy: 0.6084 - auc: 0.9207 - val_loss: 1.2396 -
val_categorical_accuracy: 0.5484 - val_auc: 0.9027
Epoch 43/150
93/93 [==============================] - 6s 66ms/step - loss: 1.1160 -
categorical_accuracy: 0.6025 - auc: 0.9212 - val_loss: 1.2893 -
val_categorical_accuracy: 0.5591 - val_auc: 0.8953
Epoch 44/150
93/93 [==============================] - 6s 66ms/step - loss: 1.1065 -
categorical_accuracy: 0.6111 - auc: 0.9230 - val_loss: 1.2437 -
val_categorical_accuracy: 0.5742 - val_auc: 0.9023
Epoch 45/150
93/93 [==============================] - 19s 204ms/step - loss: 1.1091 -
categorical_accuracy: 0.6036 - auc: 0.9220 - val_loss: 1.2376 -
val_categorical_accuracy: 0.5462 - val_auc: 0.9034
Epoch 46/150
93/93 [==============================] - 6s 66ms/step - loss: 1.1036 -
categorical_accuracy: 0.6019 - auc: 0.9231 - val_loss: 1.2387 -
val_categorical_accuracy: 0.5441 - val_auc: 0.9031
Epoch 47/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0982 -
categorical_accuracy: 0.6068 - auc: 0.9241 - val_loss: 1.2658 -
val_categorical_accuracy: 0.5505 - val_auc: 0.8995
Epoch 48/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0937 -
categorical_accuracy: 0.6100 - auc: 0.9246 - val_loss: 1.2790 -
val_categorical_accuracy: 0.5355 - val_auc: 0.8974
Epoch 49/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0895 -
categorical_accuracy: 0.6095 - auc: 0.9259 - val_loss: 1.2590 -
val_categorical_accuracy: 0.5398 - val_auc: 0.9005
Epoch 50/150
```

```
93/93 [==============================] - 19s 205ms/step - loss: 1.0882 -
categorical_accuracy: 0.6009 - auc: 0.9247 - val_loss: 1.2365 -
val_categorical_accuracy: 0.5548 - val_auc: 0.9048
Epoch 51/150
93/93 [==============================] - 19s 205ms/step - loss: 1.0881 -
categorical_accuracy: 0.6143 - auc: 0.9261 - val_loss: 1.2340 -
val_categorical_accuracy: 0.5570 - val_auc: 0.9038
Epoch 52/150
93/93 [==============================] - 19s 203ms/step - loss: 1.0828 -
categorical_accuracy: 0.6105 - auc: 0.9259 - val_loss: 1.2333 -
val_categorical_accuracy: 0.5484 - val_auc: 0.9046
Epoch 53/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0784 -
categorical_accuracy: 0.6009 - auc: 0.9269 - val_loss: 1.2361 -
val_categorical_accuracy: 0.5484 - val_auc: 0.9044
Epoch 54/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0744 -
categorical_accuracy: 0.6111 - auc: 0.9271 - val_loss: 1.2995 -
val_categorical_accuracy: 0.5462 - val_auc: 0.8953
Epoch 55/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0750 -
categorical_accuracy: 0.6057 - auc: 0.9276 - val_loss: 1.2424 -
val_categorical_accuracy: 0.5484 - val_auc: 0.9028
Epoch 56/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0686 -
categorical_accuracy: 0.6052 - auc: 0.9284 - val_loss: 1.2478 -
val_categorical_accuracy: 0.5527 - val_auc: 0.9029
Epoch 57/150
93/93 [==============================] - 19s 205ms/step - loss: 1.0677 -
categorical_accuracy: 0.6073 - auc: 0.9287 - val_loss: 1.2297 -
val_categorical_accuracy: 0.5419 - val_auc: 0.9055
Epoch 58/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0625 -
categorical_accuracy: 0.6095 - auc: 0.9292 - val_loss: 1.2469 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9034
Epoch 59/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0598 -
categorical_accuracy: 0.6170 - auc: 0.9297 - val_loss: 1.2824 -
val_categorical_accuracy: 0.5484 - val_auc: 0.8968
Epoch 60/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0577 -
categorical_accuracy: 0.6175 - auc: 0.9301 - val_loss: 1.2418 -
val_categorical_accuracy: 0.5484 - val_auc: 0.9041
Epoch 61/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0582 -
categorical_accuracy: 0.6116 - auc: 0.9302 - val_loss: 1.2911 -
val_categorical_accuracy: 0.5505 - val_auc: 0.8958
Epoch 62/150
```

```
93/93 [==============================] - 6s 66ms/step - loss: 1.0460 -
categorical_accuracy: 0.6165 - auc: 0.9314 - val_loss: 1.2400 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9052
Epoch 63/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0538 -
categorical_accuracy: 0.6105 - auc: 0.9308 - val_loss: 1.2333 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9059
Epoch 64/150
93/93 [==============================] - ETA: 0s - loss: 1.0475 -
categorical_accuracy: 0.6122 - auc: 0.9312
Epoch 00064: ReduceLROnPlateau reducing learning rate to 0.0004999999888241291.
93/93 [==============================] - 6s 66ms/step - loss: 1.0475 -
categorical_accuracy: 0.6122 - auc: 0.9312 - val_loss: 1.2490 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9031
Epoch 65/150
93/93 [==============================] - 19s 205ms/step - loss: 1.0469 -
categorical_accuracy: 0.5966 - auc: 0.9311 - val_loss: 1.2272 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9064
Epoch 66/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0219 -
categorical_accuracy: 0.6251 - auc: 0.9352 - val_loss: 1.2283 -
val_categorical_accuracy: 0.5570 - val_auc: 0.9062
Epoch 67/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0208 -
categorical_accuracy: 0.6240 - auc: 0.9355 - val_loss: 1.2359 -
val_categorical_accuracy: 0.5613 - val_auc: 0.9053
Epoch 68/150
93/93 [==============================] - 19s 206ms/step - loss: 1.0188 -
categorical_accuracy: 0.6283 - auc: 0.9359 - val_loss: 1.2254 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9071
Epoch 69/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0185 -
categorical_accuracy: 0.6315 - auc: 0.9358 - val_loss: 1.2298 -
val_categorical_accuracy: 0.5398 - val_auc: 0.9064
Epoch 70/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0166 -
categorical_accuracy: 0.6235 - auc: 0.9361 - val_loss: 1.2295 -
val_categorical_accuracy: 0.5570 - val_auc: 0.9066
Epoch 71/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0192 -
categorical_accuracy: 0.6251 - auc: 0.9357 - val_loss: 1.2324 -
val_categorical_accuracy: 0.5462 - val_auc: 0.9062
Epoch 72/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0156 -
categorical_accuracy: 0.6288 - auc: 0.9363 - val_loss: 1.2314 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9064
Epoch 73/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0168 -
```

```
categorical_accuracy: 0.6283 - auc: 0.9359 - val_loss: 1.2291 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9065
Epoch 74/150
93/93 [==============================] - 19s 202ms/step - loss: 1.0163 -
categorical_accuracy: 0.6261 - auc: 0.9360 - val_loss: 1.2245 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9068
Epoch 75/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0166 -
categorical_accuracy: 0.6304 - auc: 0.9359 - val_loss: 1.2322 -
val_categorical_accuracy: 0.5462 - val_auc: 0.9062
Epoch 76/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0183 -
categorical_accuracy: 0.6267 - auc: 0.9357 - val_loss: 1.2457 -
val_categorical_accuracy: 0.5548 - val_auc: 0.9041
Epoch 77/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0153 -
categorical_accuracy: 0.6278 - auc: 0.9360 - val_loss: 1.2332 -
val_categorical_accuracy: 0.5527 - val_auc: 0.9061
Epoch 78/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0134 -
categorical_accuracy: 0.6299 - auc: 0.9365 - val_loss: 1.2277 -
val_categorical_accuracy: 0.5527 - val_auc: 0.9066
Epoch 79/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0150 -
categorical_accuracy: 0.6261 - auc: 0.9362 - val_loss: 1.2361 -
val_categorical_accuracy: 0.5591 - val_auc: 0.9052
Epoch 80/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0159 -
categorical_accuracy: 0.6175 - auc: 0.9361 - val_loss: 1.2374 -
val_categorical_accuracy: 0.5548 - val_auc: 0.9053
Epoch 81/150
93/93 [==============================] - ETA: 0s - loss: 1.0132 -
categorical_accuracy: 0.6294 - auc: 0.9365
Epoch 00081: ReduceLROnPlateau reducing learning rate to 4.9999996554106475e-05.
93/93 [==============================] - 6s 66ms/step - loss: 1.0132 -
categorical_accuracy: 0.6294 - auc: 0.9365 - val_loss: 1.2253 -
val_categorical_accuracy: 0.5462 - val_auc: 0.9069
Epoch 82/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0145 -
categorical_accuracy: 0.6240 - auc: 0.9355 - val_loss: 1.2279 -
val_categorical_accuracy: 0.5527 - val_auc: 0.9066
Epoch 83/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0098 -
categorical_accuracy: 0.6283 - auc: 0.9371 - val_loss: 1.2291 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9065
Epoch 84/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0099 -
categorical_accuracy: 0.6294 - auc: 0.9368 - val_loss: 1.2300 -
```

```
val_categorical_accuracy: 0.5505 - val_auc: 0.9065
Epoch 00084: early stopping
```

2.2 Criando outro check point

[5]:
```
reset_keras()
```

```
499
```

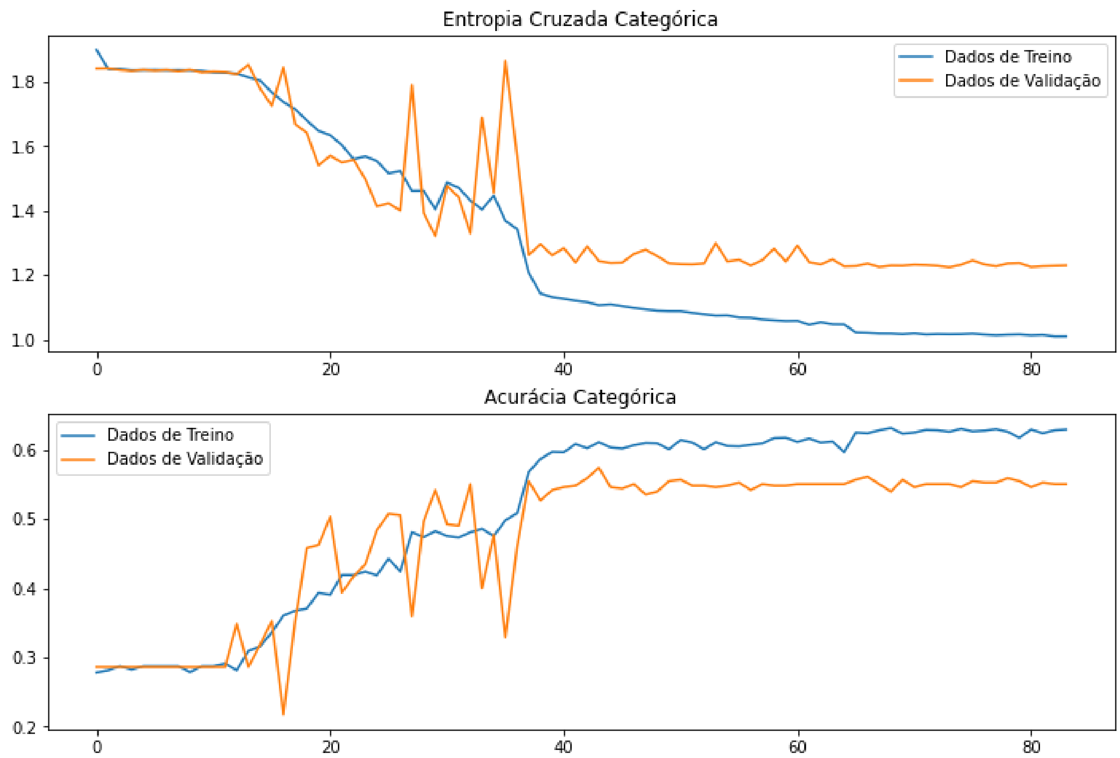
[ ]:
```
modelo1 = load_model("./modelos/modelo1.hdf5")
modelo1_history = json.load(open("./modelos/history_modelo1.json",'r'))
```

[6]:
```
tfidf = pickle.load(open("tfidf.plk","rb"))
tf_treino = pickle.load(open("tf_treino.plk","rb"))
idf_treino = pickle.load(open("idf_treino.plk","rb"))
DocFreq = pickle.load(open("DocFreq.plk","rb"))
bag_words_corpus = pickle.load(open("bag_of_words.plk","rb"))
listagem = pickle.load(open("listagem.pkl","rb"))
dicionario = pickle.load(open("dicionario.pkl","rb"))

df_training = pd.read_csv("df_training.csv")

N = len(bag_words_corpus)

x_train,x_test,x_valid,y_train,y_test,y_valid =␣
 ↪train_test_valid_split(df_treino= df_training)
```

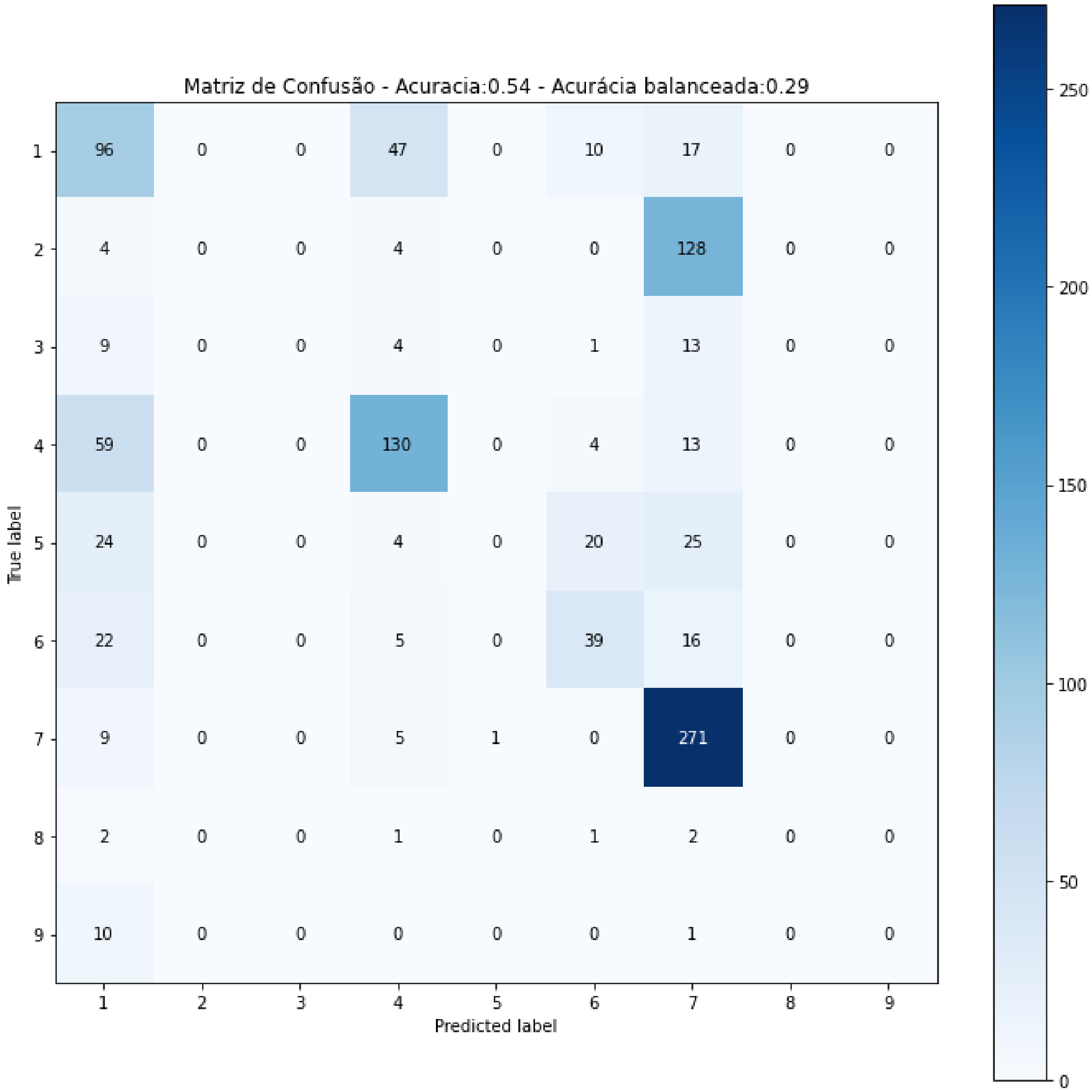
[24]:
```
X,X_train,X_test,X_valid,Y_train,Y_test,Y_valid =␣
 ↪dados_modelo_treino_valid(x_train,x_test,x_valid,df_training,dicionario,tfidf,N)
```

2.3 Verificando o Modelo 1

[7]:
```
plot_treinamento(modelo1_history)
```



[22]:
```
#Há muitos problemas com a classe 2, 5, 8 e 9 (estas últimas pela quantidade de
↪amostras)
plot_matriz_confusao(modelo1,X_test,Y_test)
```



[10]:
```
balanced_accuracy_score(y_pred=tf.argmax(modelo1.predict(X_test), axis = 1)+1,
                                         y_true=Y_test.argmax(axis=1)+1)
```

[10]: 0.2909928941297776

[11]:
```
accuracy_score(y_pred=tf.argmax(modelo1.predict(X_test), axis = 1)+1,
                                         y_true=Y_test.argmax(axis=1)+1)
```

[11]: 0.5376128385155466

2.3.1 Tentativa de melhora do modelo - remover palavras que são comuns a todas as classes - Modelo 2

[7]:
```
palavras_comuns =␣
 ↪['fig','et','al','mutat','cell','activ','0','1','2','3','4','5','6','7','8','9']
palavras_comuns
```

[7]:
```
['fig',
 'et',
 'al',
 'mutat',
 'cell',
 'activ',
 '0',
 '1',
 '2',
 '3',
 '4',
 '5',
 '6',
 '7',
 '8',
 '9']
```

[8]:
```
listagem_nova,bag_of_words_novo,dicionario_novo,N_novo,DocFreq_novo,tf_novo,idf_novo,tfidf_nov
 ↪= remove_palavras(listagem,
                                                                                        ␣
 ↪                                  bag_words_corpus,
                                                                                        ␣
 ↪                                  palavras_comuns)
```

[76]:
```
#Verificando a nuvem de palavras, usando TF-IDF, com a nova listagem
plota_wordcloud(word_cloud_plot_tfidf,
                df_training,tfidf_novo)
```

[72]:
```python
pickle.dump(tfidf_novo,open("tfidf_novo.plk","wb"))
pickle.dump(tf_novo,open("tf_novo.plk","wb"))
pickle.dump(idf_novo,open("idf_novo.plk","wb"))
pickle.dump(DocFreq_novo,open("DocFreq_novo.plk","wb"))
pickle.dump(bag_of_words_novo,open("bag_of_words_novo.plk","wb"))
pickle.dump(listagem_nova,open("listagem_nova.pkl","wb"))
pickle.dump(dicionario_novo,open("dicionario_novo.pkl","wb"))
```

[16]:
```python
tfidf_novo=pickle.load(open("tfidf_novo.plk","rb"))
#tf_novo=pickle.load(open("tf_novo.plk","rb"))
#idf_novo = pickle.load(open("idf_novo.plk","rb"))
DocFreq_novo = pickle.load(open("DocFreq_novo.plk","rb"))
bag_of_words_novo = pickle.load(open("bag_of_words_novo.plk","rb"))
listagem_nova=pickle.load(open("listagem_nova.pkl","rb"))
dicionario_novo=pickle.load(open("dicionario_novo.pkl","rb"))
```

[7]:
```python
# Gerando segundo modelo
N_novo = len(bag_of_words_novo)
X_novo,X_train_novo,X_test_novo,X_valid_novo,Y_train_novo,Y_test_novo,Y_valid_novo␣
↪ = dados_modelo_treino_valid(x_train,x_test,x_valid,
                                                                              ␣
↪                                    df_training,dicionario_novo,
                                                                              ␣
↪                                    tfidf_novo,N_novo)
```

[8]:
```python
reset_keras()
```

477

[12]:
```
camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name =␣
↪"Camada_Entrada")

primeira_camada_oculta = Dense(1000,activation = 'relu',kernel_initializer =␣
↪'uniform',name = "Camada_Oculta_1")(camada_entrada)

segunda_camada_oculta = Dense(100,activation = 'relu',kernel_initializer =␣
↪'uniform',name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer =␣
↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo2 = Model(inputs = [camada_entrada], outputs = [camada_saida])
```

[17]:
```
modelo2.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])
```

[18]:
```
modelo2 =␣
↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
               saving_checkpoint_path="./modelos/", nome_modelo="modelo2",␣
↪modelo= modelo2)
```

```
Epoch 1/150
93/93 [==============================] - 20s 217ms/step - loss: 1.8990 -
categorical_accuracy: 0.2797 - auc: 0.7413 - val_loss: 1.8328 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7661
Epoch 2/150
93/93 [==============================] - 19s 204ms/step - loss: 1.8349 -
categorical_accuracy: 0.2867 - auc: 0.7579 - val_loss: 1.8258 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7810
Epoch 3/150
93/93 [==============================] - 19s 205ms/step - loss: 1.8046 -
categorical_accuracy: 0.3093 - auc: 0.7718 - val_loss: 1.7480 -
val_categorical_accuracy: 0.2903 - val_auc: 0.8082
Epoch 4/150
93/93 [==============================] - 19s 201ms/step - loss: 1.6240 -
categorical_accuracy: 0.4217 - auc: 0.8295 - val_loss: 1.4885 -
val_categorical_accuracy: 0.4581 - val_auc: 0.8646
Epoch 5/150
93/93 [==============================] - 19s 204ms/step - loss: 1.4845 -
categorical_accuracy: 0.4508 - auc: 0.8585 - val_loss: 1.4049 -
val_categorical_accuracy: 0.4882 - val_auc: 0.8733
Epoch 6/150
93/93 [==============================] - 19s 204ms/step - loss: 1.3364 -
categorical_accuracy: 0.4987 - auc: 0.8864 - val_loss: 1.3322 -
val_categorical_accuracy: 0.5011 - val_auc: 0.8881
Epoch 7/150
```

```
93/93 [==============================] - 19s 204ms/step - loss: 1.2695 -
categorical_accuracy: 0.5433 - auc: 0.8971 - val_loss: 1.2515 -
val_categorical_accuracy: 0.5462 - val_auc: 0.9008
Epoch 8/150
93/93 [==============================] - 19s 206ms/step - loss: 1.1823 -
categorical_accuracy: 0.5573 - auc: 0.9114 - val_loss: 1.2504 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9014
Epoch 9/150
93/93 [==============================] - 19s 202ms/step - loss: 1.1696 -
categorical_accuracy: 0.5750 - auc: 0.9135 - val_loss: 1.2065 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9091
Epoch 10/150
93/93 [==============================] - 6s 66ms/step - loss: 1.0891 -
categorical_accuracy: 0.6095 - auc: 0.9252 - val_loss: 1.2471 -
val_categorical_accuracy: 0.5376 - val_auc: 0.9014
Epoch 11/150
93/93 [==============================] - 19s 204ms/step - loss: 1.0535 -
categorical_accuracy: 0.6073 - auc: 0.9300 - val_loss: 1.1645 -
val_categorical_accuracy: 0.5742 - val_auc: 0.9144
Epoch 12/150
93/93 [==============================] - 6s 66ms/step - loss: 0.9853 -
categorical_accuracy: 0.6294 - auc: 0.9384 - val_loss: 1.1749 -
val_categorical_accuracy: 0.5634 - val_auc: 0.9127
Epoch 13/150
93/93 [==============================] - 6s 66ms/step - loss: 0.9472 -
categorical_accuracy: 0.6493 - auc: 0.9436 - val_loss: 1.1707 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9120
Epoch 14/150
93/93 [==============================] - 6s 66ms/step - loss: 0.9162 -
categorical_accuracy: 0.6547 - auc: 0.9473 - val_loss: 2.0446 -
val_categorical_accuracy: 0.3269 - val_auc: 0.7959
Epoch 15/150
93/93 [==============================] - 19s 204ms/step - loss: 0.9166 -
categorical_accuracy: 0.6595 - auc: 0.9469 - val_loss: 1.1269 -
val_categorical_accuracy: 0.5763 - val_auc: 0.9186
Epoch 16/150
93/93 [==============================] - 6s 66ms/step - loss: 0.8360 -
categorical_accuracy: 0.6815 - auc: 0.9558 - val_loss: 1.1416 -
val_categorical_accuracy: 0.5978 - val_auc: 0.9177
Epoch 17/150
93/93 [==============================] - 19s 203ms/step - loss: 0.8696 -
categorical_accuracy: 0.6649 - auc: 0.9522 - val_loss: 1.1160 -
val_categorical_accuracy: 0.5957 - val_auc: 0.9224
Epoch 18/150
93/93 [==============================] - 6s 66ms/step - loss: 0.7929 -
categorical_accuracy: 0.7074 - auc: 0.9599 - val_loss: 1.3368 -
val_categorical_accuracy: 0.5161 - val_auc: 0.8954
Epoch 19/150
```

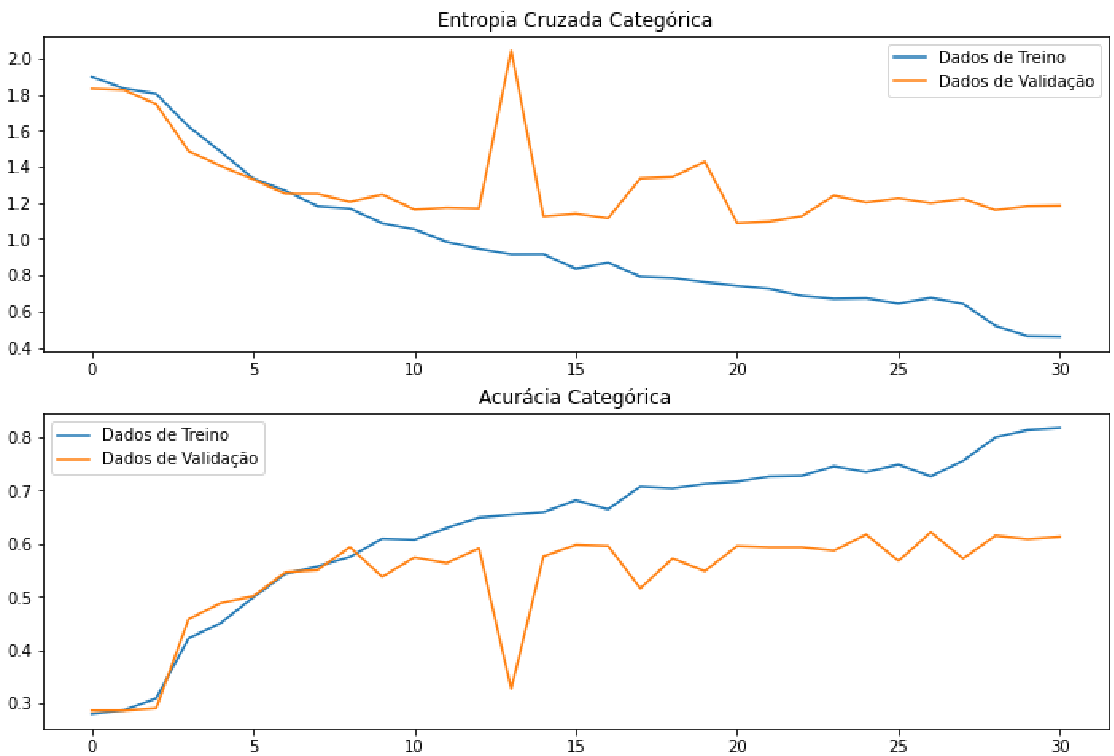
```
93/93 [==============================] - 6s 66ms/step - loss: 0.7858 -
categorical_accuracy: 0.7041 - auc: 0.9603 - val_loss: 1.3450 -
val_categorical_accuracy: 0.5720 - val_auc: 0.8960
Epoch 20/150
93/93 [==============================] - 6s 66ms/step - loss: 0.7620 -
categorical_accuracy: 0.7127 - auc: 0.9623 - val_loss: 1.4289 -
val_categorical_accuracy: 0.5484 - val_auc: 0.8862
Epoch 21/150
93/93 [==============================] - 19s 204ms/step - loss: 0.7424 -
categorical_accuracy: 0.7171 - auc: 0.9647 - val_loss: 1.0895 -
val_categorical_accuracy: 0.5957 - val_auc: 0.9246
Epoch 22/150
93/93 [==============================] - 6s 66ms/step - loss: 0.7256 -
categorical_accuracy: 0.7267 - auc: 0.9658 - val_loss: 1.0983 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9259
Epoch 23/150
93/93 [==============================] - 6s 66ms/step - loss: 0.6871 -
categorical_accuracy: 0.7278 - auc: 0.9699 - val_loss: 1.1274 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9228
Epoch 24/150
93/93 [==============================] - 6s 66ms/step - loss: 0.6707 -
categorical_accuracy: 0.7456 - auc: 0.9709 - val_loss: 1.2416 -
val_categorical_accuracy: 0.5871 - val_auc: 0.9143
Epoch 25/150
93/93 [==============================] - 6s 66ms/step - loss: 0.6739 -
categorical_accuracy: 0.7348 - auc: 0.9708 - val_loss: 1.2029 -
val_categorical_accuracy: 0.6172 - val_auc: 0.9118
Epoch 26/150
93/93 [==============================] - 6s 66ms/step - loss: 0.6443 -
categorical_accuracy: 0.7488 - auc: 0.9735 - val_loss: 1.2266 -
val_categorical_accuracy: 0.5677 - val_auc: 0.9167
Epoch 27/150
93/93 [==============================] - 6s 66ms/step - loss: 0.6764 -
categorical_accuracy: 0.7267 - auc: 0.9698 - val_loss: 1.1994 -
val_categorical_accuracy: 0.6215 - val_auc: 0.9090
Epoch 28/150
93/93 [==============================] - ETA: 0s - loss: 0.6429 -
categorical_accuracy: 0.7558 - auc: 0.9733
Epoch 00028: ReduceLROnPlateau reducing learning rate to 0.005000000074505806.
93/93 [==============================] - 6s 66ms/step - loss: 0.6429 -
categorical_accuracy: 0.7558 - auc: 0.9733 - val_loss: 1.2236 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9081
Epoch 29/150
93/93 [==============================] - 6s 66ms/step - loss: 0.5209 -
categorical_accuracy: 0.7999 - auc: 0.9832 - val_loss: 1.1616 -
val_categorical_accuracy: 0.6151 - val_auc: 0.9204
Epoch 30/150
93/93 [==============================] - 6s 66ms/step - loss: 0.4639 -
```

```
categorical_accuracy: 0.8139 - auc: 0.9865 - val_loss: 1.1830 -
val_categorical_accuracy: 0.6086 - val_auc: 0.9206
Epoch 31/150
93/93 [==============================] - 6s 66ms/step - loss: 0.4615 -
categorical_accuracy: 0.8176 - auc: 0.9862 - val_loss: 1.1855 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9229
Epoch 00031: early stopping
```

2.4 Verificando o modelo 2

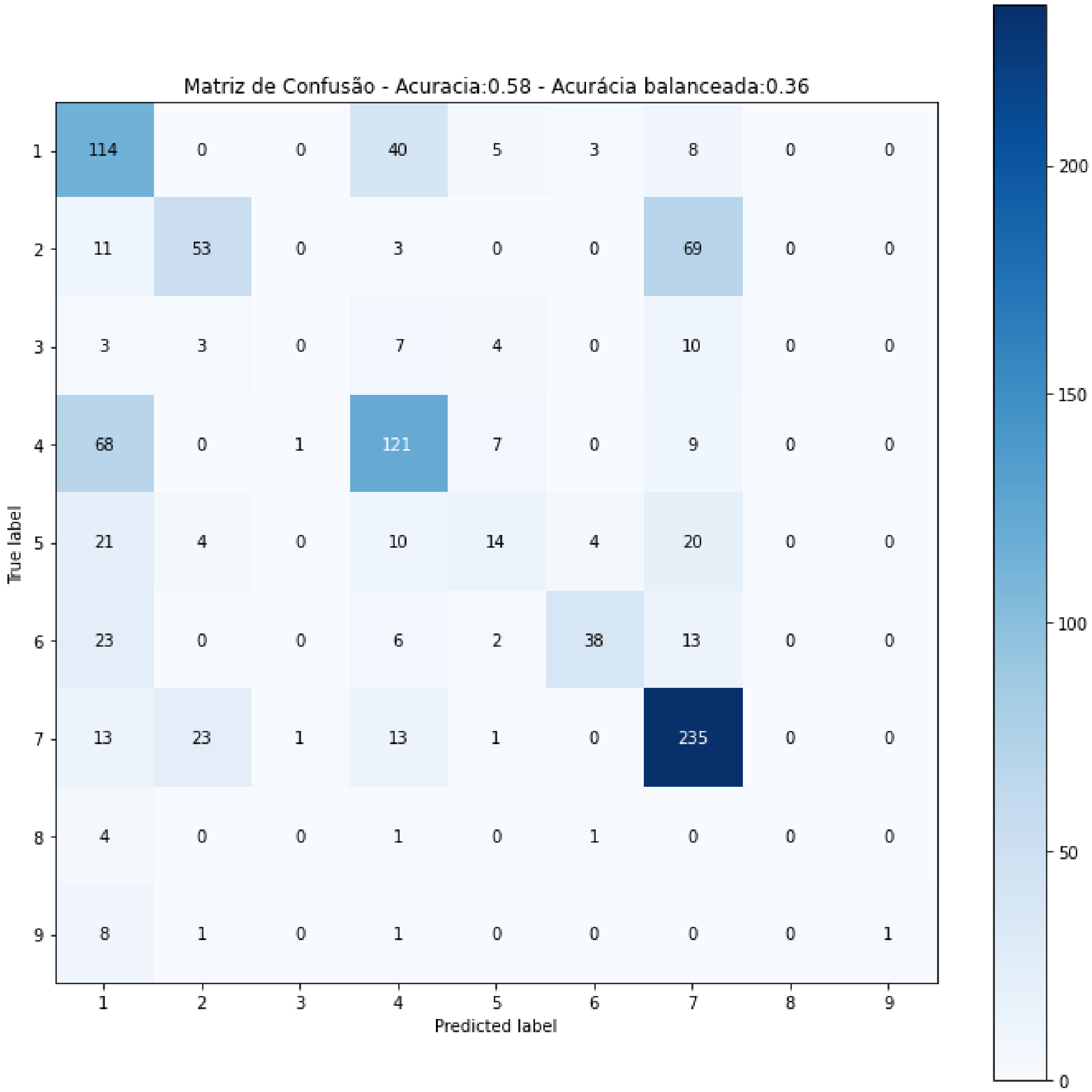
[9]:
```python
modelo2 = load_model("./modelos/modelo2.hdf5")
modelo2_history = json.load(open("./modelos/history_modelo2.json",'r'))
```

[10]:
```python
plot_treinamento(modelo2_history)
```



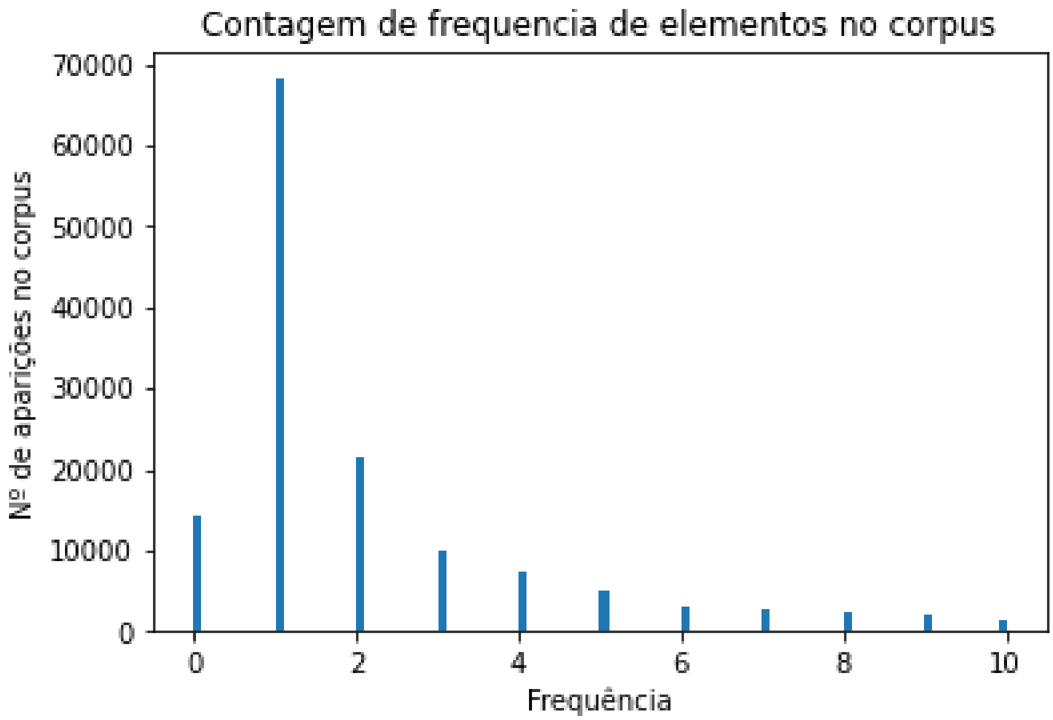
Nota-se que houve uma melhora no resultado do modelo

[11]:
```python
plot_matriz_confusao(modelo2,X_test_novo,Y_test_novo)
```



2.5 Tentativa de melhora do modelo - remover palavras com pouca frequencia - Modelo 3

[29]:
```
contagem_frequencia = np.array([valor for chave,valor in DocFreq_novo.items()])
plt.hist(contagem_frequencia[contagem_frequencia <= 10], bins = 100)
plt.title("Contagem de frequencia de elementos no corpus")
plt.xlabel("Frequência")
plt.ylabel("Nž de aparições no corpus")
plt.show()
```



[30]:
```
palavras_remover2 = [chave for chave,valor in DocFreq_novo.items() if valor <=␣
 ↪1]
len(palavras_remover2)
```

[30]: 82493

[34]:
```
listagem_nova,bag_of_words_novo,dicionario_novo,N_novo,DocFreq_novo,tf_novo,idf_novo,tfidf_nov
 ↪= remove_palavras(listagem_nova,
                                                                                ␣
 ↪                                   bag_of_words_novo,
                                                                                ␣
 ↪                                   palavras_remover2)
```

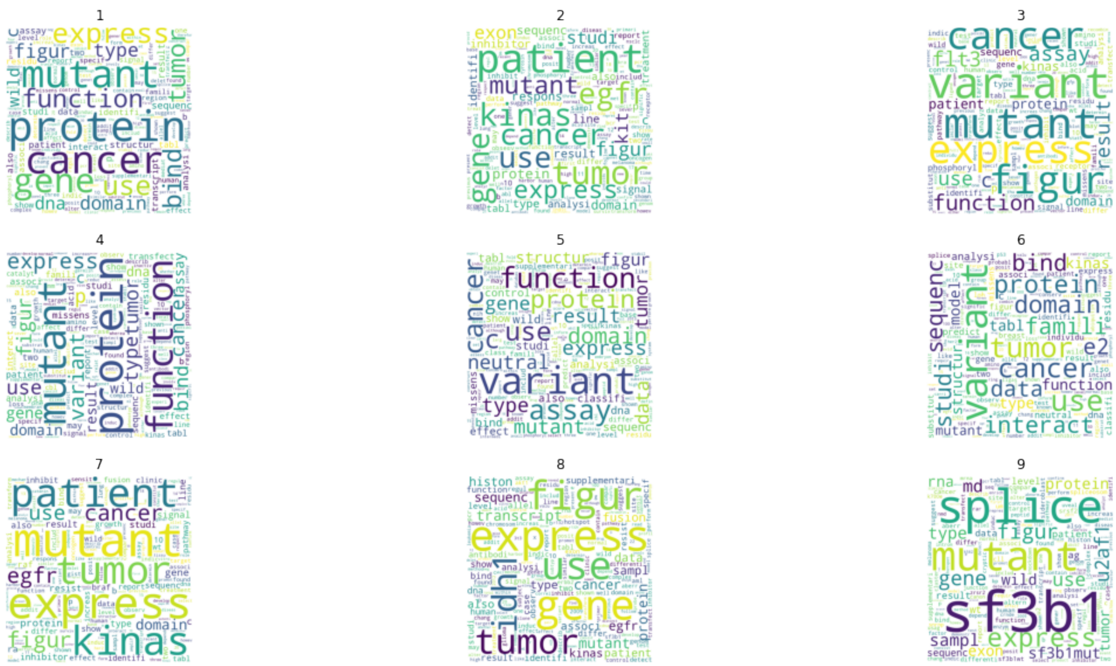
```
Listagem nova concluida
BOW concluído
Dicionário novo concluído
DoqFreq novo concluído
TF e IDF novos concluídos
TFIDF novo concluído
```

[27]:
```
pickle.dump(tfidf_novo,open("tfidf_novo.plk","wb"))
pickle.dump(tf_novo,open("tf_novo.plk","wb"))
pickle.dump(idf_novo,open("idf_novo.plk","wb"))
pickle.dump(DocFreq_novo,open("DocFreq_novo.plk","wb"))
pickle.dump(bag_of_words_novo,open("bag_of_words_novo.plk","wb"))
```

```
pickle.dump(listagem_nova,open("listagem_nova.pkl","wb"))
pickle.dump(dicionario_novo,open("dicionario_novo.pkl","wb"))
```

[7]:
```
tfidf_novo=pickle.load(open("tfidf_novo.plk","rb"))
#tf_novo=pickle.load(open("tf_novo.plk","rb"))
#idf_novo = pickle.load(open("idf_novo.plk","rb"))
DocFreq_novo = pickle.load(open("DocFreq_novo.plk","rb"))
bag_of_words_novo = pickle.load(open("bag_of_words_novo.plk","rb"))
listagem_nova=pickle.load(open("listagem_nova.pkl","rb"))
dicionario_novo=pickle.load(open("dicionario_novo.pkl","rb"))
```

[98]:
```
#Verificando a nuvem de palavras, usando TF-IDF, com a nova listagem
plota_wordcloud(word_cloud_plot_tfidf,
                df_training,tfidf_novo)
```



[ ]:
```
# Gerando terceiro modelo
N_novo = len(bag_of_words_novo)
X_novo,X_train_novo,X_test_novo,X_valid_novo,Y_train_novo,Y_test_novo,Y_valid_novo␣
↪ = dados_modelo_treino_valid(x_train,x_test,x_valid,
                                                                                     ␣
↪                                   df_training,dicionario_novo,
                                                                                     ␣
↪                                   tfidf_novo,N_novo)
```

[100]:
```
reset_keras()
```

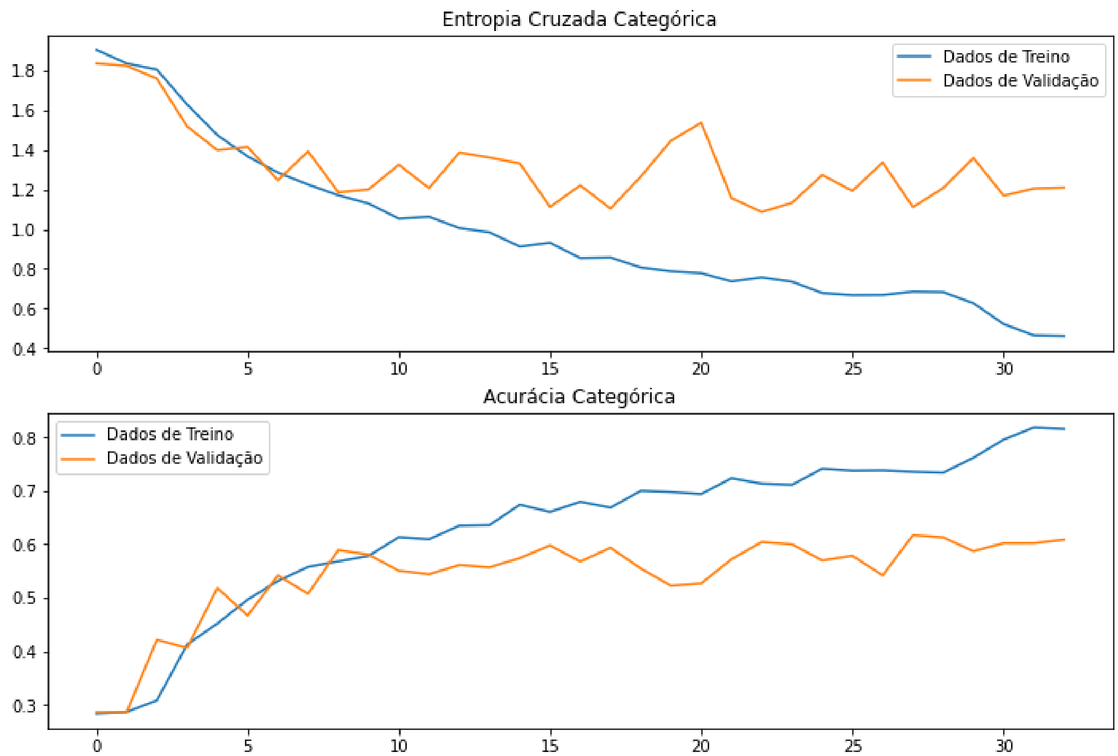
41079

[30]:
```
camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name =␣
↪"Camada_Entrada")

primeira_camada_oculta = Dense(1000,activation = 'relu',kernel_initializer =␣
↪'uniform',name = "Camada_Oculta_1")(camada_entrada)

segunda_camada_oculta = Dense(100,activation = 'relu',kernel_initializer =␣
↪'uniform',name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer =␣
↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo3 = Model(inputs = [camada_entrada], outputs = [camada_saida])

modelo3.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])

modelo3 =␣
↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
               saving_checkpoint_path="./modelos/", nome_modelo="modelo3",␣
↪modelo= modelo3)
```

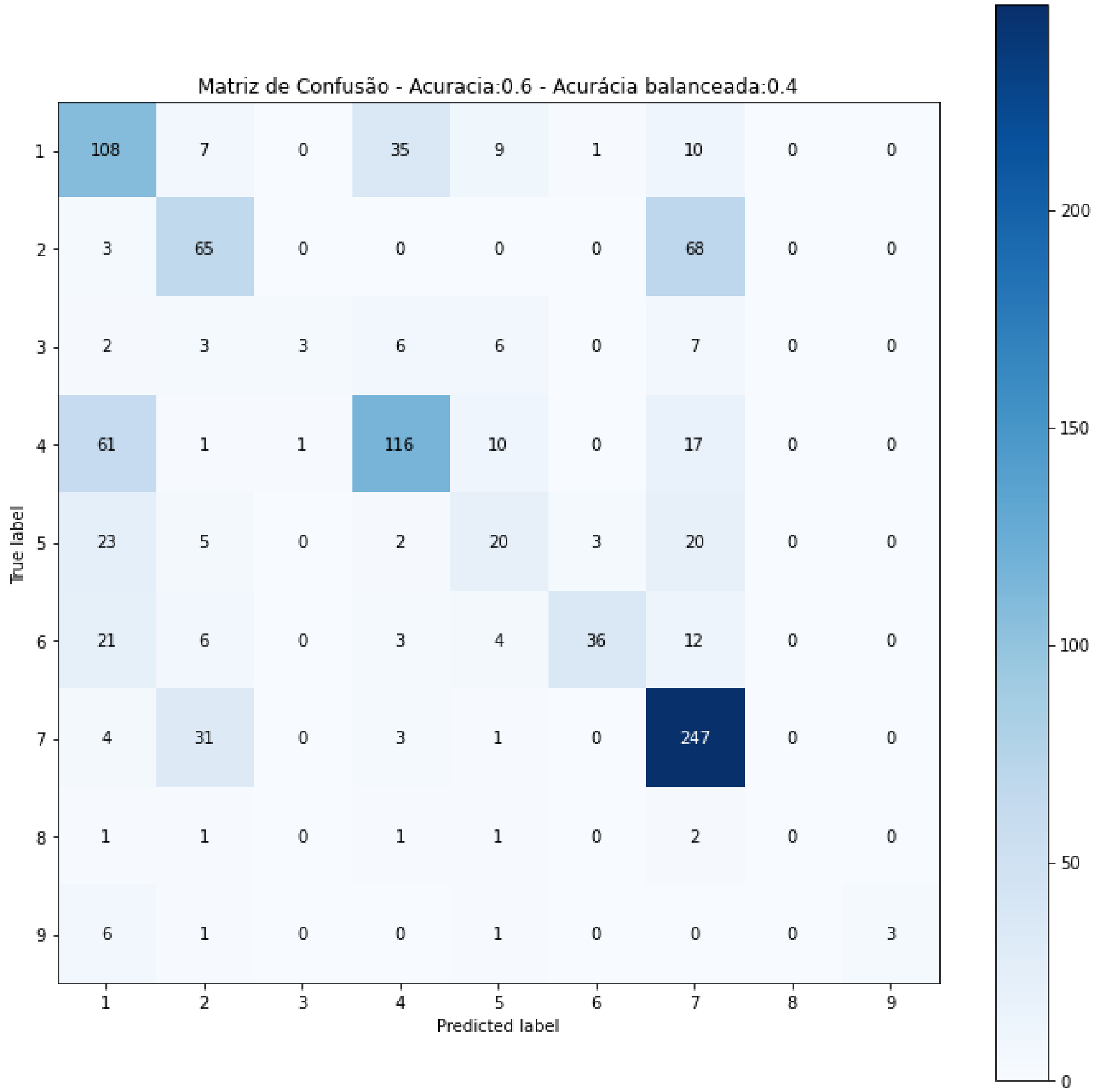
2.6 Verificando o modelo 3

[47]:
```
modelo3 = load_model("./modelos/modelo3.hdf5")
```

[48]:
```
modelo3_history = json.load(open("./modelos/history_modelo3.json",'r'))
plot_treinamento(modelo3_history)
```

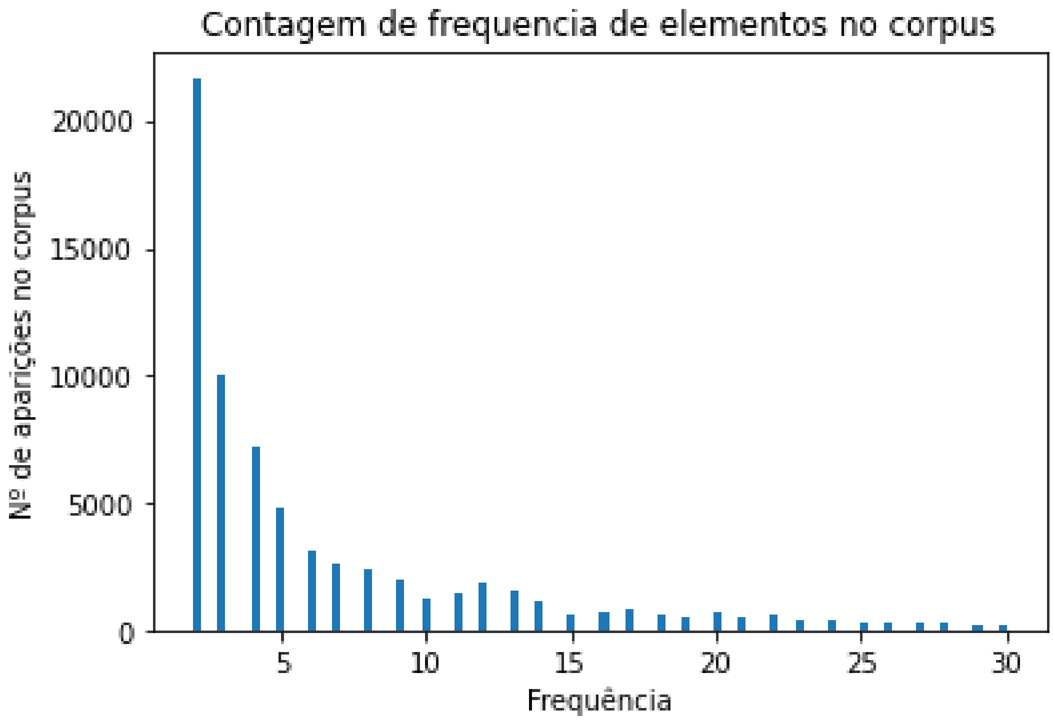


[49]:
```
plot_matriz_confusao(modelo3,X_test_novo,Y_test_novo)
```



Removendo palavras com 2 ocorrencias - Modelo 4

[57]:
```
contagem_frequencia = np.array([valor for chave,valor in DocFreq_novo.items()])
plt.hist(contagem_frequencia[contagem_frequencia <= 30], bins = 100)
plt.title("Contagem de frequencia de elementos no corpus")
plt.xlabel("Frequência")
plt.ylabel("Nž de aparições no corpus")
plt.show()
```



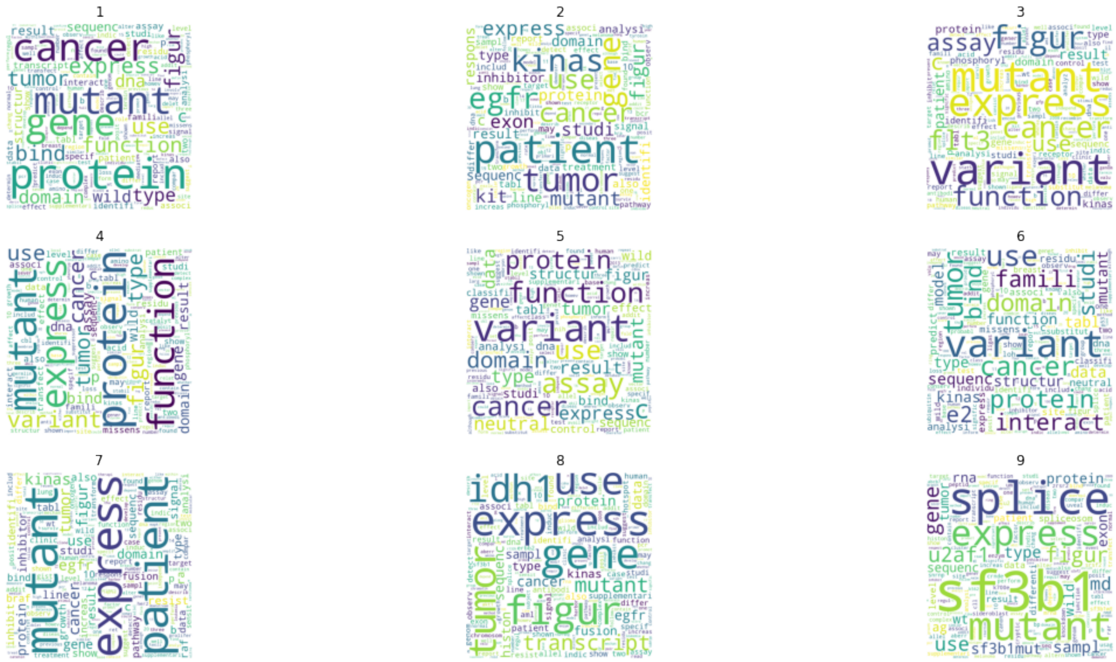
[64]:
```
palavras_remover3 = [chave for chave,valor in DocFreq_novo.items() if valor <=␣
 ↪2]
len(palavras_remover3)
```

[64]: 21605

[65]:
```
listagem_nova,bag_of_words_novo,dicionario_novo,N_novo,DocFreq_novo,tf_novo,idf_novo,tfidf_nov
 ↪= remove_palavras(listagem_nova,
                                                                                ␣
 ↪                                   bag_of_words_novo,
                                                                                ␣
 ↪                                   palavras_remover3)
```

```
Conjunto de palavras a remover concluído
Listagem nova concluida
BOW concluído
Dicionário novo concluído
DoqFreq novo concluído
TF e IDF novos concluídos
TFIDF novo concluído
```

[66]:
```
#Verificando a nuvem de palavras, usando TF-IDF, com a nova listagem
plota_wordcloud(word_cloud_plot_tfidf,
                df_training,tfidf_novo)
```



[11]:
```
# Gerando quarto modelo
N_novo = len(bag_of_words_novo)
X_novo,X_train_novo,X_test_novo,X_valid_novo,Y_train_novo,Y_test_novo,Y_valid_novo = dados_modelo_treino_valid(x_train,x_test,x_valid,
                                    df_training,dicionario_novo,
                                    tfidf_novo,N_novo)
```

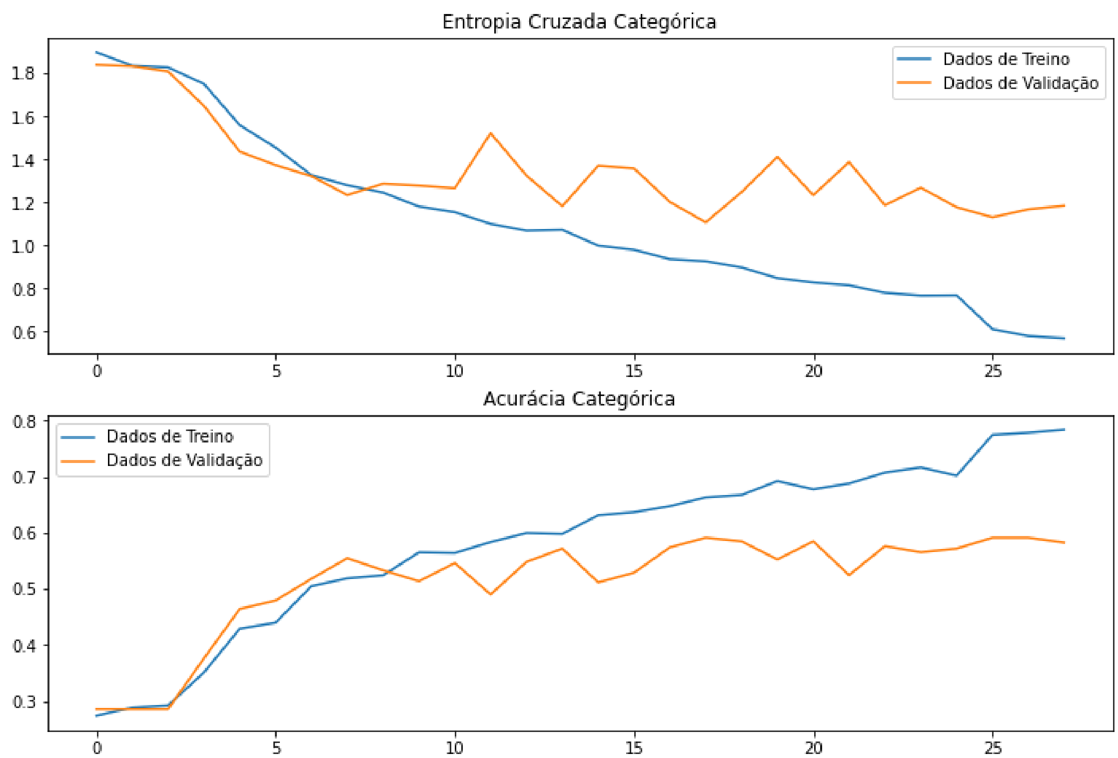
[8]:
```
reset_keras()

camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name = "Camada_Entrada")

primeira_camada_oculta = Dense(1000,activation = 'relu',kernel_initializer = 'uniform',name = "Camada_Oculta_1")(camada_entrada)

segunda_camada_oculta = Dense(100,activation = 'relu',kernel_initializer = 'uniform',name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer = 'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo4 = Model(inputs = [camada_entrada], outputs = [camada_saida])

modelo4.compile(loss = 'categorical_crossentropy',
```

```
                 optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                 metrics = ['categorical_accuracy',AUC()])

modelo4 =␣
↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
                saving_checkpoint_path="./modelos/", nome_modelo="modelo4",␣
↪modelo= modelo4)
```

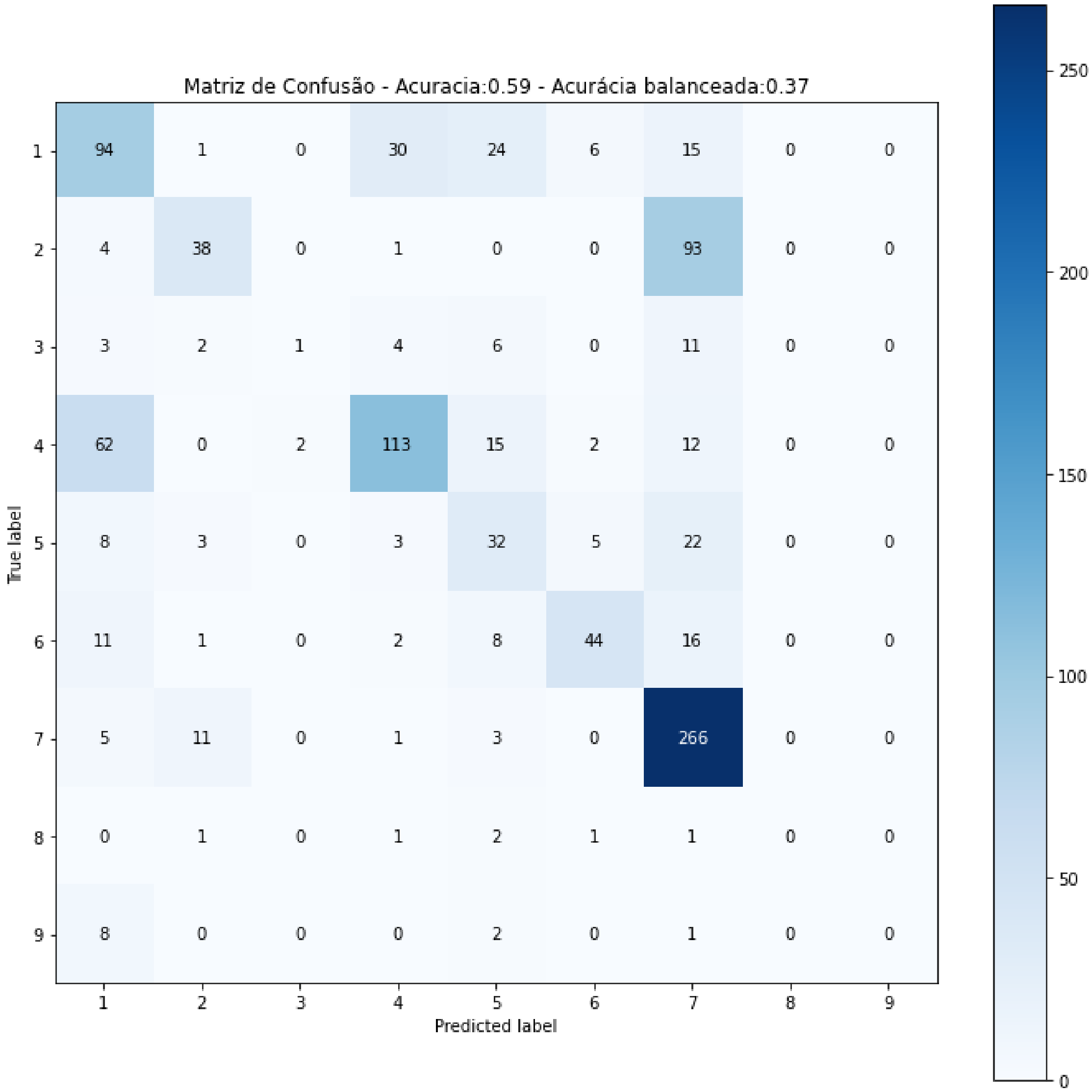
2.7 Verificando o modelo 4

[5]:
```
modelo4 = load_model("./modelos/modelo4.hdf5")
modelo4_history = json.load(open("./modelos/history_modelo4.json",'r'))
```

[6]:
```
plot_treinamento(modelo4_history)
```



[75]:
```
plot_matriz_confusao(modelo4,X_test_novo,Y_test_novo)
```



O modelo 4 não apresentou melhoras, comparado ao modelo 3

2.8 Tentativa de melhora do modelo - inserindo palavras que faltam que existem no dataframe - Modelo 5

Algumas palavras, que estão no dataset de treino, não estão no Texto

[53]:
```
corpus_variation = [corpusnization(token) for token in df_training.Variation.
 ↪tolist()]
corpus_gene = [corpusnization(token) for token in df_training.Gene.tolist()]
```

[ ]:
```
gene_faltante = []
variation_faltante = []
for (i,[gene,variation]) in enumerate(zip(corpus_gene,corpus_variation)):
    for cada_gene in gene:
```

```
        try:
            DocFreq[cada_gene]
```

2.9 Criação de bigramas - uma nova abordagem

Vamos criar bigramas e verificar se eles são mais eficazes ao modelo

[73]:
```
metricas_bigramas = BigramAssocMeasures()
```

[10]:
```
colecao_bigramas = [BigramCollocationFinder.from_words(doc) for doc in␣
 ↪treino_corpus]
```

[37]:
```
#Vamos filtrar a colecao de bigramas para remover bigramas que contenha fora da␣
 ↪listagem
for doc in colecao_bigramas:
    doc.apply_ngram_filter(lambda p1,p2: False if((p1 in listagem_nova) & (p2␣
 ↪in listagem_nova)) else True)
```

[39]:
```
pickle.dump(colecao_bigramas,open("bigramas_filtrada.plk","wb"))
```

[4]:
```
colecao_bigramas=pickle.load(open("bigramas_filtrada.plk","rb"))
```

[64]:
```
for doc in colecao_bigramas:
    doc.apply_freq_filter(min_freq=10)
    doc.apply_ngram_filter(lambda p1,p2: True if((p1 == "wild") & (p2 ==␣
 ↪"type")) else False)
```

[65]:
```
list_frequencia_bigrama = []
for doc in colecao_bigramas:
    dict_aux = {}
    for chave,valor in dict(doc.ngram_fd).items():
        dict_aux.update({"".join((chave[0],"_",chave[1])): valor})
    list_frequencia_bigrama.append(dict_aux)
```

[66]:
```
plota_wordcloud(word_cloud_plot,
                df_training,list_frequencia_bigrama)
```

[90]:
```
N_novo = len(list_frequencia_bigrama)
X_novo,X_train_novo,X_test_novo,X_valid_novo,Y_train_novo,Y_test_novo,Y_valid_novo␣
↪ = dados_modelo_treino_valid(x_train,                                              ␣
↪                                  x_test,                                          ␣
↪                                  x_valid,                                         ␣
↪                                  df_training,                                     ␣
↪                                  cria_dicionario_palavras(␣
↪cria_listagem_palavras(list_frequencia_bigrama)),                                  ␣
↪                                  list_frequencia_bigrama,                         ␣
↪                                  N_novo)
```

[94]:
```
reset_keras()

camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name =␣
↪"Camada_Entrada")

primeira_camada_oculta = Dense(1000,activation = 'relu',kernel_initializer =␣
↪'uniform',name = "Camada_Oculta_1")(camada_entrada)
```

```
segunda_camada_oculta = Dense(100,activation = 'relu',kernel_initializer =␣
 ↪'uniform',name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer =␣
 ↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo5 = Model(inputs = [camada_entrada], outputs = [camada_saida])

modelo5.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])

modelo5 =␣
 ↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
               saving_checkpoint_path="./modelos/", nome_modelo="modelo5",␣
 ↪modelo= modelo5)
```

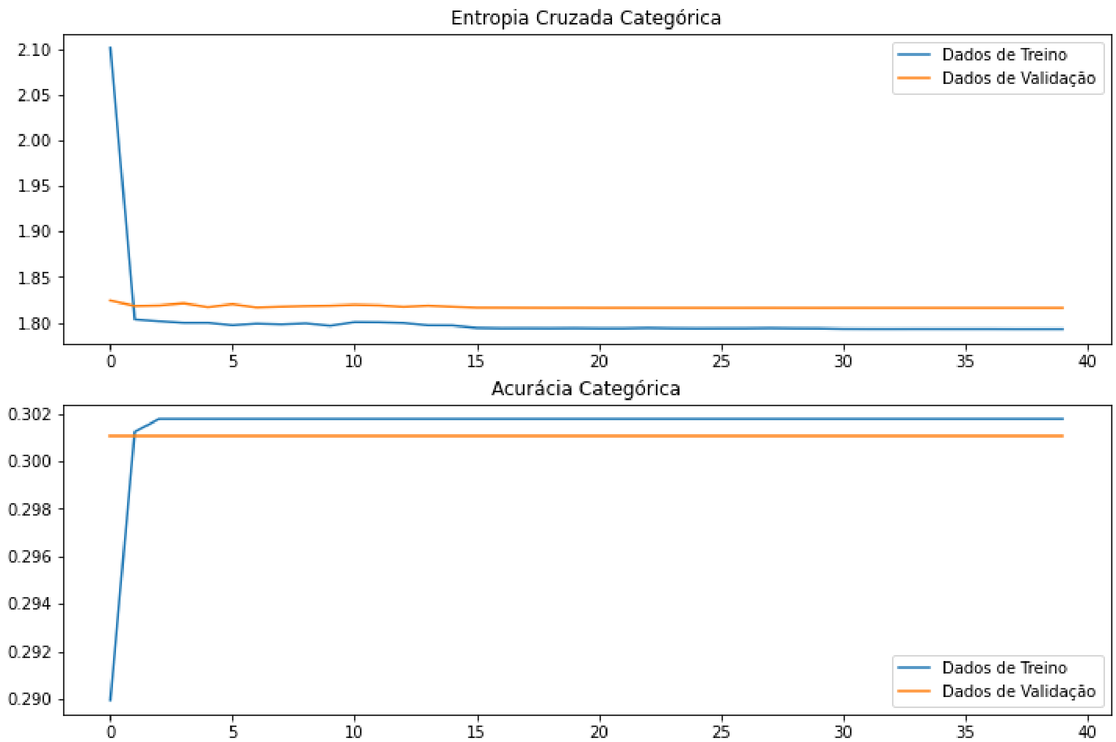
```
19073
Epoch 1/150
93/93 [==============================] - 7s 78ms/step - loss: 2.1013 -
categorical_accuracy: 0.2899 - auc: 0.7374 - val_loss: 1.8241 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 2/150
93/93 [==============================] - 4s 41ms/step - loss: 1.8035 -
categorical_accuracy: 0.3012 - auc: 0.7664 - val_loss: 1.8183 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7680
Epoch 3/150
93/93 [==============================] - 3s 27ms/step - loss: 1.8014 -
categorical_accuracy: 0.3018 - auc: 0.7657 - val_loss: 1.8188 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 4/150
93/93 [==============================] - 3s 29ms/step - loss: 1.7997 -
categorical_accuracy: 0.3018 - auc: 0.7663 - val_loss: 1.8212 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 5/150
93/93 [==============================] - 3s 36ms/step - loss: 1.7997 -
categorical_accuracy: 0.3018 - auc: 0.7671 - val_loss: 1.8170 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 6/150
93/93 [==============================] - 3s 27ms/step - loss: 1.7969 -
categorical_accuracy: 0.3018 - auc: 0.7674 - val_loss: 1.8202 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7641
Epoch 7/150
93/93 [==============================] - 4s 41ms/step - loss: 1.7988 -
categorical_accuracy: 0.3018 - auc: 0.7670 - val_loss: 1.8166 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 8/150
```

```
93/93 [==============================] - 2s 24ms/step - loss: 1.7979 -
categorical_accuracy: 0.3018 - auc: 0.7661 - val_loss: 1.8176 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7680
Epoch 9/150
93/93 [==============================] - 3s 28ms/step - loss: 1.7991 -
categorical_accuracy: 0.3018 - auc: 0.7667 - val_loss: 1.8182 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7680
Epoch 10/150
93/93 [==============================] - 3s 30ms/step - loss: 1.7964 -
categorical_accuracy: 0.3018 - auc: 0.7674 - val_loss: 1.8186 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 11/150
93/93 [==============================] - 3s 30ms/step - loss: 1.8005 -
categorical_accuracy: 0.3018 - auc: 0.7664 - val_loss: 1.8195 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 12/150
93/93 [==============================] - 3s 31ms/step - loss: 1.8003 -
categorical_accuracy: 0.3018 - auc: 0.7665 - val_loss: 1.8190 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7641
Epoch 13/150
93/93 [==============================] - 2s 23ms/step - loss: 1.7995 -
categorical_accuracy: 0.3018 - auc: 0.7665 - val_loss: 1.8174 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 14/150
91/93 [============================>.] - ETA: 0s - loss: 1.7978 -
categorical_accuracy: 0.3000 - auc: 0.7674
Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.005000000074505806.
93/93 [==============================] - 2s 23ms/step - loss: 1.7969 -
categorical_accuracy: 0.3018 - auc: 0.7677 - val_loss: 1.8186 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7680
Epoch 15/150
93/93 [==============================] - 3s 27ms/step - loss: 1.7968 -
categorical_accuracy: 0.3018 - auc: 0.7691 - val_loss: 1.8175 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7680
Epoch 16/150
93/93 [==============================] - 4s 42ms/step - loss: 1.7941 -
categorical_accuracy: 0.3018 - auc: 0.7680 - val_loss: 1.8164 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 17/150
93/93 [==============================] - 4s 39ms/step - loss: 1.7937 -
categorical_accuracy: 0.3018 - auc: 0.7679 - val_loss: 1.8164 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 18/150
93/93 [==============================] - 4s 45ms/step - loss: 1.7938 -
categorical_accuracy: 0.3018 - auc: 0.7680 - val_loss: 1.8163 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 19/150
93/93 [==============================] - 4s 40ms/step - loss: 1.7937 -
```

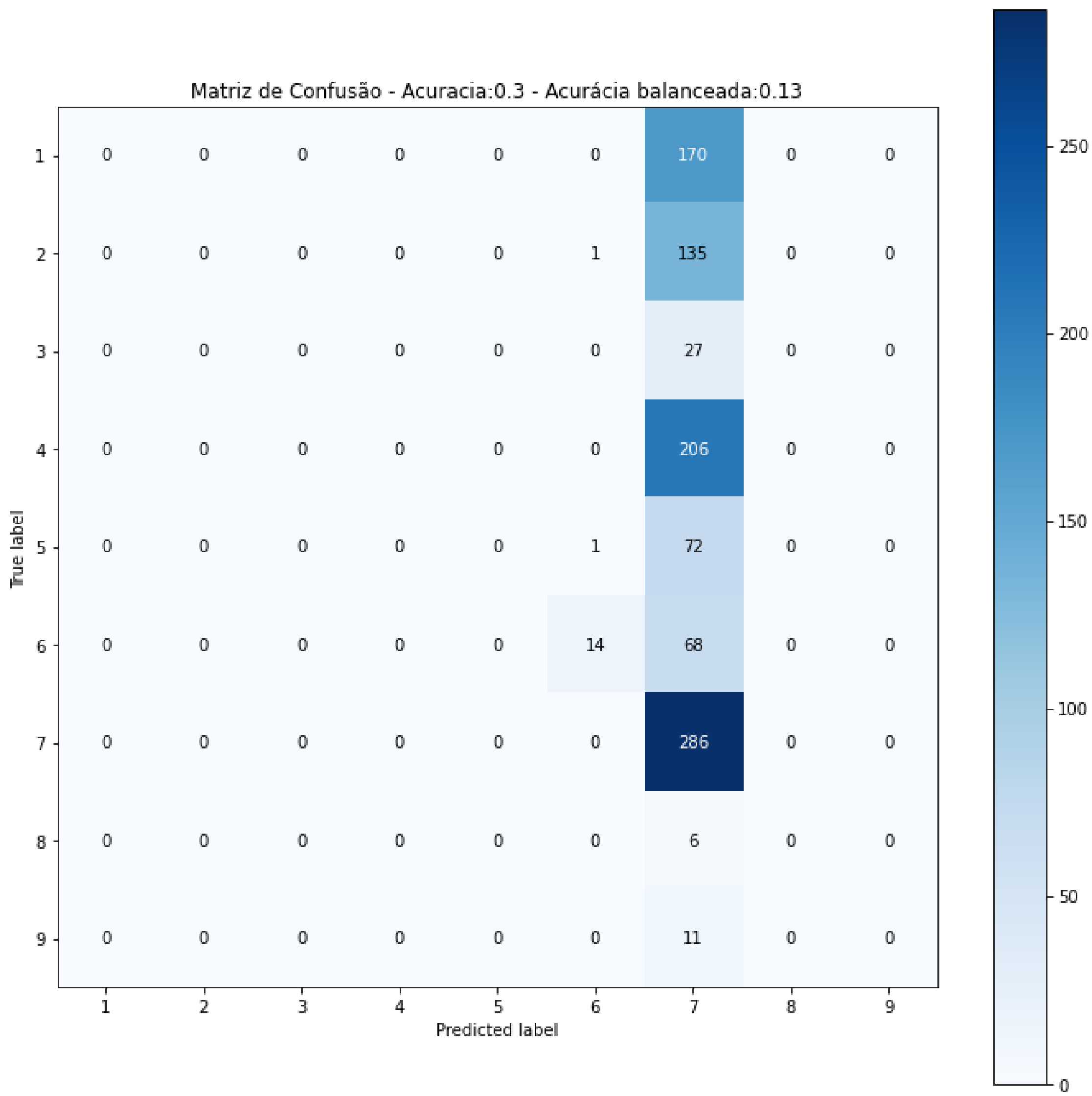
```
categorical_accuracy: 0.3018 - auc: 0.7683 - val_loss: 1.8163 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7682
Epoch 20/150
93/93 [==============================] - 3s 29ms/step - loss: 1.7938 -
categorical_accuracy: 0.3018 - auc: 0.7677 - val_loss: 1.8163 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 21/150
93/93 [==============================] - 4s 43ms/step - loss: 1.7936 -
categorical_accuracy: 0.3018 - auc: 0.7686 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 22/150
93/93 [==============================] - 2s 24ms/step - loss: 1.7936 -
categorical_accuracy: 0.3018 - auc: 0.7682 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 23/150
93/93 [==============================] - 4s 43ms/step - loss: 1.7941 -
categorical_accuracy: 0.3018 - auc: 0.7676 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 24/150
93/93 [==============================] - 4s 41ms/step - loss: 1.7937 -
categorical_accuracy: 0.3018 - auc: 0.7685 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 25/150
93/93 [==============================] - 3s 27ms/step - loss: 1.7936 -
categorical_accuracy: 0.3018 - auc: 0.7678 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 26/150
93/93 [==============================] - 4s 38ms/step - loss: 1.7937 -
categorical_accuracy: 0.3018 - auc: 0.7684 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 27/150
93/93 [==============================] - 3s 30ms/step - loss: 1.7937 -
categorical_accuracy: 0.3018 - auc: 0.7686 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 28/150
93/93 [==============================] - 2s 26ms/step - loss: 1.7939 -
categorical_accuracy: 0.3018 - auc: 0.7683 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 29/150
93/93 [==============================] - 4s 44ms/step - loss: 1.7937 -
categorical_accuracy: 0.3018 - auc: 0.7676 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 30/150
89/93 [===========================>..] - ETA: 0s - loss: 1.7962 -
categorical_accuracy: 0.3006 - auc: 0.7672
Epoch 00030: ReduceLROnPlateau reducing learning rate to 0.0004999999888241291.
93/93 [==============================] - 4s 43ms/step - loss: 1.7936 -
categorical_accuracy: 0.3018 - auc: 0.7683 - val_loss: 1.8162 -
```

```
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 31/150
93/93 [==============================] - 2s 25ms/step - loss: 1.7931 -
categorical_accuracy: 0.3018 - auc: 0.7691 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 32/150
93/93 [==============================] - 3s 29ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7686 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 33/150
93/93 [==============================] - 3s 27ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7691 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 34/150
93/93 [==============================] - 3s 29ms/step - loss: 1.7931 -
categorical_accuracy: 0.3018 - auc: 0.7690 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 35/150
93/93 [==============================] - 3s 30ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7686 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 36/150
93/93 [==============================] - 3s 28ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7685 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 37/150
91/93 [============================>.] - ETA: 0s - loss: 1.7924 -
categorical_accuracy: 0.3033 - auc: 0.7695
Epoch 00037: ReduceLROnPlateau reducing learning rate to 4.9999996554106475e-05.
93/93 [==============================] - 3s 31ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7693 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 38/150
93/93 [==============================] - 3s 31ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7693 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 39/150
93/93 [==============================] - 2s 26ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7693 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 40/150
93/93 [==============================] - 2s 27ms/step - loss: 1.7930 -
categorical_accuracy: 0.3018 - auc: 0.7693 - val_loss: 1.8162 -
val_categorical_accuracy: 0.3011 - val_auc: 0.7683
Epoch 00040: early stopping
```

[95]:
```
modelo5_history = json.load(open("./modelos/history_modelo5.json",'r'))
plot_treinamento(modelo5_history)
```



[96]:
```
plot_matriz_confusao(modelo5,X_test_novo,Y_test_novo)
```



Este foi o pior modelo até o momento

[80]:
```python
#Tentando usar PMI
list_frequencia_bigrama_PMI = []
for doc in colecao_bigramas:
    dict_aux = {}
    for chave,valor in dict(doc.score_ngrams(metricas_bigramas.pmi)).items():
        dict_aux.update({"".join((chave[0],"_",chave[1])): valor})
    list_frequencia_bigrama_PMI.append(dict_aux)
```

[ ]:
```python
X_novo,X_train_novo,X_test_novo,X_valid_novo,Y_train_novo,Y_test_novo,Y_valid_novo␣
↪ = dados_modelo_treino_valid(x_train,
                                                                            ␣
↪                                        x_test,
```

```
                                          x_valid,
                                          df_training,
                                          cria_dicionario_palavras(
↪cria_listagem_palavras(list_frequencia_bigrama_PMI)),
                                          list_frequencia_bigrama_PMI,
                                          N_novo)
```

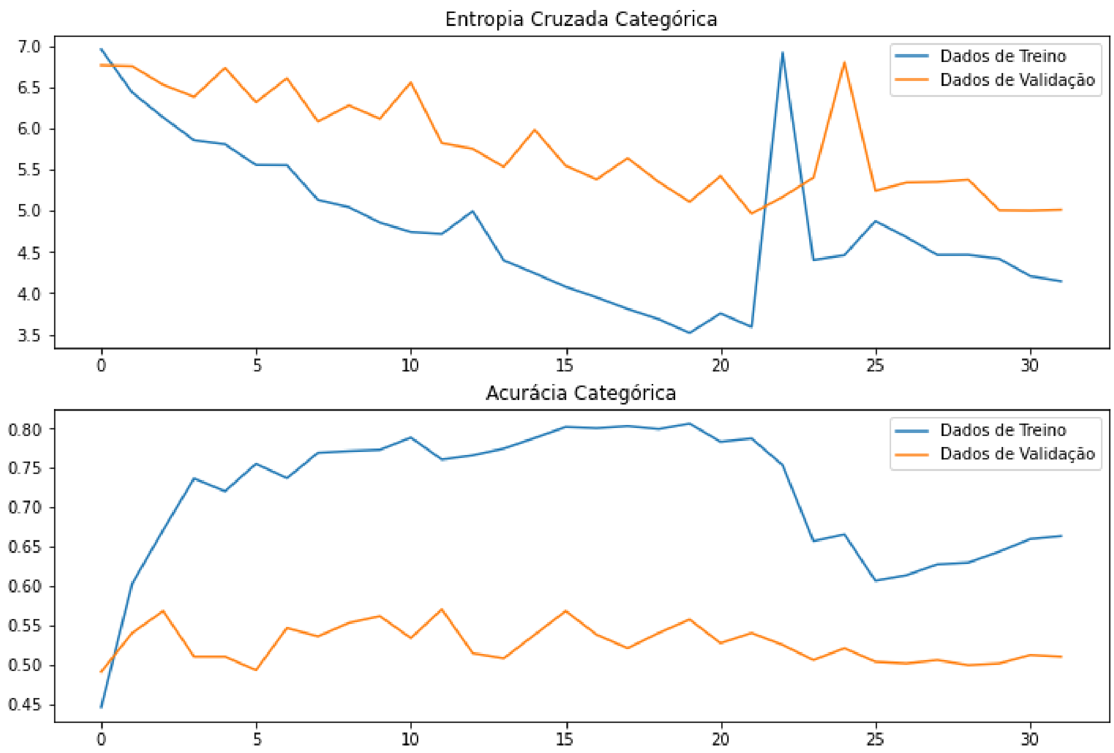
[87]:
```
reset_keras()

camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name =
 ↪"Camada_Entrada")

primeira_camada_oculta = Dense(1000,activation = 'relu',kernel_initializer =
 ↪'uniform',
                               kernel_regularizer = regularizers.l1_l2(l1=1e-5,
 ↪l2=1e-4),
                               bias_regularizer=regularizers.l2(1e-4),
                               activity_regularizer=regularizers.l2(1e-5),
                               name = "Camada_Oculta_1")(camada_entrada)

segunda_camada_oculta = Dense(100,activation = 'relu',kernel_initializer =
 ↪'uniform',
                              kernel_regularizer = regularizers.l1_l2(l1=1e-5,
 ↪l2=1e-4),
                              bias_regularizer=regularizers.l2(1e-4),
                              activity_regularizer=regularizers.l2(1e-5),
                              name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer =
 ↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo6 = Model(inputs = [camada_entrada], outputs = [camada_saida])

modelo6.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])

modelo6 =
 ↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
               saving_checkpoint_path="./modelos/", nome_modelo="modelo6",
 ↪modelo= modelo6)
```

```
346
Epoch 1/150
93/93 [==============================] - 11s 119ms/step - loss: 6.9546 -
categorical_accuracy: 0.4454 - auc: 0.8443 - val_loss: 6.7617 -
val_categorical_accuracy: 0.4903 - val_auc: 0.8694
Epoch 2/150
93/93 [==============================] - 8s 82ms/step - loss: 6.4326 -
categorical_accuracy: 0.6025 - auc: 0.9178 - val_loss: 6.7511 -
val_categorical_accuracy: 0.5398 - val_auc: 0.8662
Epoch 3/150
93/93 [==============================] - 7s 80ms/step - loss: 6.1302 -
categorical_accuracy: 0.6708 - auc: 0.9372 - val_loss: 6.5257 -
val_categorical_accuracy: 0.5677 - val_auc: 0.8770
Epoch 4/150
93/93 [==============================] - 8s 83ms/step - loss: 5.8538 -
categorical_accuracy: 0.7364 - auc: 0.9529 - val_loss: 6.3793 -
val_categorical_accuracy: 0.5097 - val_auc: 0.8806
Epoch 5/150
93/93 [==============================] - 6s 65ms/step - loss: 5.8042 -
categorical_accuracy: 0.7203 - auc: 0.9494 - val_loss: 6.7296 -
val_categorical_accuracy: 0.5097 - val_auc: 0.8537
Epoch 6/150
93/93 [==============================] - 7s 79ms/step - loss: 5.5529 -
categorical_accuracy: 0.7552 - auc: 0.9616 - val_loss: 6.3133 -
val_categorical_accuracy: 0.4925 - val_auc: 0.8598
Epoch 7/150
93/93 [==============================] - 6s 67ms/step - loss: 5.5502 -
categorical_accuracy: 0.7370 - auc: 0.9572 - val_loss: 6.6046 -
val_categorical_accuracy: 0.5462 - val_auc: 0.8598
Epoch 8/150
93/93 [==============================] - 7s 78ms/step - loss: 5.1304 -
categorical_accuracy: 0.7692 - auc: 0.9713 - val_loss: 6.0804 -
val_categorical_accuracy: 0.5355 - val_auc: 0.8636
Epoch 9/150
93/93 [==============================] - 6s 66ms/step - loss: 5.0426 -
categorical_accuracy: 0.7714 - auc: 0.9705 - val_loss: 6.2766 -
val_categorical_accuracy: 0.5527 - val_auc: 0.8649
Epoch 10/150
93/93 [==============================] - 6s 67ms/step - loss: 4.8557 -
categorical_accuracy: 0.7730 - auc: 0.9727 - val_loss: 6.1135 -
val_categorical_accuracy: 0.5613 - val_auc: 0.8656
Epoch 11/150
93/93 [==============================] - 6s 63ms/step - loss: 4.7417 -
categorical_accuracy: 0.7886 - auc: 0.9733 - val_loss: 6.5541 -
val_categorical_accuracy: 0.5333 - val_auc: 0.8481
Epoch 12/150
93/93 [==============================] - 7s 79ms/step - loss: 4.7175 -
categorical_accuracy: 0.7606 - auc: 0.9669 - val_loss: 5.8185 -
```

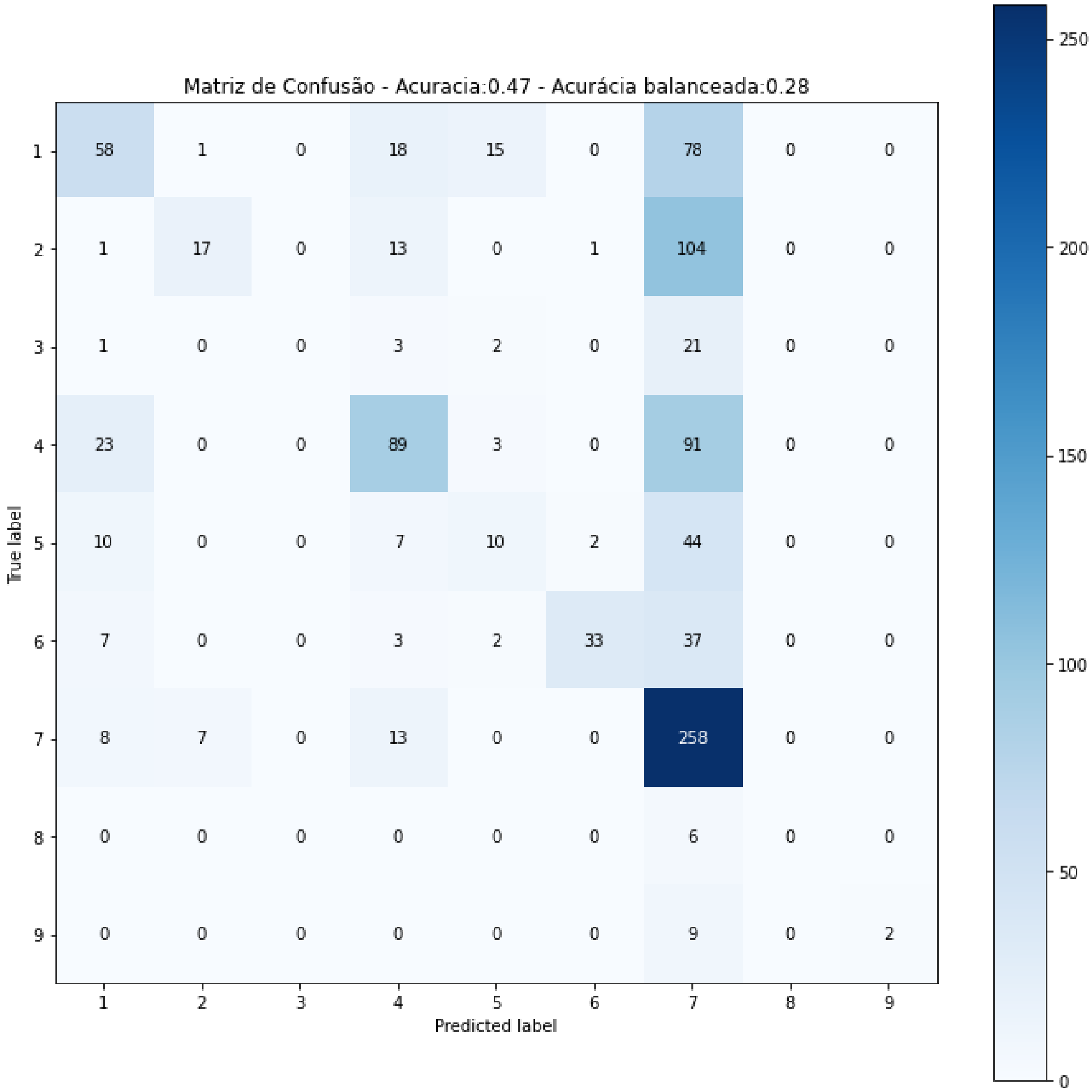
```
val_categorical_accuracy: 0.5699 - val_auc: 0.8651
Epoch 13/150
93/93 [==============================] - 8s 84ms/step - loss: 4.9945 -
categorical_accuracy: 0.7660 - auc: 0.9666 - val_loss: 5.7501 -
val_categorical_accuracy: 0.5140 - val_auc: 0.8506
Epoch 14/150
93/93 [==============================] - 7s 78ms/step - loss: 4.3981 -
categorical_accuracy: 0.7746 - auc: 0.9737 - val_loss: 5.5296 -
val_categorical_accuracy: 0.5075 - val_auc: 0.8669
Epoch 15/150
93/93 [==============================] - 6s 69ms/step - loss: 4.2418 -
categorical_accuracy: 0.7881 - auc: 0.9759 - val_loss: 5.9799 -
val_categorical_accuracy: 0.5376 - val_auc: 0.8422
Epoch 16/150
93/93 [==============================] - 6s 66ms/step - loss: 4.0773 -
categorical_accuracy: 0.8020 - auc: 0.9792 - val_loss: 5.5445 -
val_categorical_accuracy: 0.5677 - val_auc: 0.8547
Epoch 17/150
93/93 [==============================] - 8s 90ms/step - loss: 3.9498 -
categorical_accuracy: 0.8004 - auc: 0.9798 - val_loss: 5.3796 -
val_categorical_accuracy: 0.5376 - val_auc: 0.8592
Epoch 18/150
93/93 [==============================] - 6s 65ms/step - loss: 3.8091 -
categorical_accuracy: 0.8031 - auc: 0.9798 - val_loss: 5.6363 -
val_categorical_accuracy: 0.5204 - val_auc: 0.8453
Epoch 19/150
93/93 [==============================] - 7s 79ms/step - loss: 3.6856 -
categorical_accuracy: 0.7994 - auc: 0.9800 - val_loss: 5.3494 -
val_categorical_accuracy: 0.5398 - val_auc: 0.8539
Epoch 20/150
93/93 [==============================] - 8s 81ms/step - loss: 3.5198 -
categorical_accuracy: 0.8063 - auc: 0.9833 - val_loss: 5.1070 -
val_categorical_accuracy: 0.5570 - val_auc: 0.8604
Epoch 21/150
93/93 [==============================] - 6s 67ms/step - loss: 3.7549 -
categorical_accuracy: 0.7832 - auc: 0.9747 - val_loss: 5.4191 -
val_categorical_accuracy: 0.5269 - val_auc: 0.8340
Epoch 22/150
93/93 [==============================] - 7s 80ms/step - loss: 3.5921 -
categorical_accuracy: 0.7875 - auc: 0.9742 - val_loss: 4.9651 -
val_categorical_accuracy: 0.5398 - val_auc: 0.8603
Epoch 23/150
93/93 [==============================] - 6s 63ms/step - loss: 6.9151 -
categorical_accuracy: 0.7536 - auc: 0.9596 - val_loss: 5.1645 -
val_categorical_accuracy: 0.5247 - val_auc: 0.8375
Epoch 24/150
93/93 [==============================] - 6s 69ms/step - loss: 4.4026 -
categorical_accuracy: 0.6568 - auc: 0.9219 - val_loss: 5.4020 -
```

```
val_categorical_accuracy: 0.5054 - val_auc: 0.8310
Epoch 25/150
93/93 [==============================] - 7s 73ms/step - loss: 4.4617 -
categorical_accuracy: 0.6654 - auc: 0.9280 - val_loss: 6.7975 -
val_categorical_accuracy: 0.5204 - val_auc: 0.8169
Epoch 26/150
93/93 [==============================] - 6s 65ms/step - loss: 4.8741 -
categorical_accuracy: 0.6062 - auc: 0.9104 - val_loss: 5.2406 -
val_categorical_accuracy: 0.5032 - val_auc: 0.8249
Epoch 27/150
93/93 [==============================] - 6s 66ms/step - loss: 4.6792 -
categorical_accuracy: 0.6132 - auc: 0.9108 - val_loss: 5.3414 -
val_categorical_accuracy: 0.5011 - val_auc: 0.8417
Epoch 28/150
93/93 [==============================] - 4s 46ms/step - loss: 4.4646 -
categorical_accuracy: 0.6272 - auc: 0.9223 - val_loss: 5.3491 -
val_categorical_accuracy: 0.5054 - val_auc: 0.8377
Epoch 29/150
93/93 [==============================] - ETA: 0s - loss: 4.4654 -
categorical_accuracy: 0.6294 - auc: 0.9175 ETA: 0s - loss: 4.4981 - categorica
Epoch 00029: ReduceLROnPlateau reducing learning rate to 0.005000000074505806.
93/93 [==============================] - 4s 43ms/step - loss: 4.4654 -
categorical_accuracy: 0.6294 - auc: 0.9175 - val_loss: 5.3773 -
val_categorical_accuracy: 0.4989 - val_auc: 0.8329
Epoch 30/150
93/93 [==============================] - 4s 48ms/step - loss: 4.4176 -
categorical_accuracy: 0.6434 - auc: 0.9214 - val_loss: 5.0058 -
val_categorical_accuracy: 0.5011 - val_auc: 0.8475
Epoch 31/150
93/93 [==============================] - 4s 43ms/step - loss: 4.2077 -
categorical_accuracy: 0.6595 - auc: 0.9316 - val_loss: 5.0020 -
val_categorical_accuracy: 0.5118 - val_auc: 0.8504
Epoch 32/150
93/93 [==============================] - 4s 41ms/step - loss: 4.1454 -
categorical_accuracy: 0.6633 - auc: 0.9359 - val_loss: 5.0127 -
val_categorical_accuracy: 0.5097 - val_auc: 0.8490
Epoch 00032: early stopping
```

[88]:
```
modelo6_history = json.load(open("./modelos/history_modelo6.json",'r'))
plot_treinamento(modelo6_history)
```



[89]:
```
plot_matriz_confusao(modelo6,X_test_novo,Y_test_novo)
```



2.10 Pegando o melhor modelo (modelo 3) e verificando se, alterando a estrutura da rede e aplicando regularização, podemos obter melhorias

1 - incluindo regularização no modelo e reduzindo a quantidade de neuronios na camada de entrada Houve uma necessidade de redução de neurônios por conta da limitação do hardware

[110]:
```
reset_keras()

camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name =␣
↪"Camada_Entrada")

primeira_camada_oculta = Dense(500,activation = 'relu',kernel_initializer =␣
↪'uniform',
```

```
                                kernel_regularizer = regularizers.l1_l2(l1=1e-5,␣
↪l2=1e-4),
                                bias_regularizer=regularizers.l2(1e-4),
                                activity_regularizer=regularizers.l2(1e-5),
                                name = "Camada_Oculta_1")(camada_entrada)

segunda_camada_oculta = Dense(100,activation = 'relu',kernel_initializer =␣
↪'uniform',
                               kernel_regularizer = regularizers.l1_l2(l1=1e-5,␣
↪l2=1e-4),
                               bias_regularizer=regularizers.l2(1e-4),
                               activity_regularizer=regularizers.l2(1e-5),
                               name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer =␣
↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo3_1 = Model(inputs = [camada_entrada], outputs = [camada_saida])

modelo3_1.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])

modelo3_1 =␣
↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
               saving_checkpoint_path="./modelos/", nome_modelo="modelo3_1",␣
↪modelo= modelo3_1)
```

```
107
Epoch 1/150
93/93 [==============================] - 14s 146ms/step - loss: 15.3951 -
categorical_accuracy: 0.2803 - auc: 0.7404 - val_loss: 15.1153 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7577
Epoch 2/150
93/93 [==============================] - 10s 111ms/step - loss: 14.9029 -
categorical_accuracy: 0.2803 - auc: 0.7559 - val_loss: 14.6782 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7686
Epoch 3/150
93/93 [==============================] - 10s 113ms/step - loss: 14.4691 -
categorical_accuracy: 0.2878 - auc: 0.7586 - val_loss: 14.2508 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7790
Epoch 4/150
93/93 [==============================] - 10s 110ms/step - loss: 14.0381 -
categorical_accuracy: 0.2985 - auc: 0.7654 - val_loss: 13.7924 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7890
Epoch 5/150
93/93 [==============================] - 10s 113ms/step - loss: 13.5015 -
```

```
categorical_accuracy: 0.3808 - auc: 0.8121 - val_loss: 13.1636 -
val_categorical_accuracy: 0.4495 - val_auc: 0.8525
Epoch 6/150
93/93 [==============================] - 11s 117ms/step - loss: 12.9426 -
categorical_accuracy: 0.4293 - auc: 0.8504 - val_loss: 12.6897 -
val_categorical_accuracy: 0.4387 - val_auc: 0.8609
Epoch 7/150
93/93 [==============================] - 11s 113ms/step - loss: 12.4533 -
categorical_accuracy: 0.4578 - auc: 0.8709 - val_loss: 12.2366 -
val_categorical_accuracy: 0.4710 - val_auc: 0.8737
Epoch 8/150
93/93 [==============================] - 11s 115ms/step - loss: 11.9939 -
categorical_accuracy: 0.5013 - auc: 0.8863 - val_loss: 11.8283 -
val_categorical_accuracy: 0.4946 - val_auc: 0.8829
Epoch 9/150
93/93 [==============================] - 10s 108ms/step - loss: 11.5664 -
categorical_accuracy: 0.5153 - auc: 0.8965 - val_loss: 11.5178 -
val_categorical_accuracy: 0.4968 - val_auc: 0.8751
Epoch 10/150
93/93 [==============================] - 10s 103ms/step - loss: 11.1797 -
categorical_accuracy: 0.5342 - auc: 0.9018 - val_loss: 11.0072 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9032
Epoch 11/150
93/93 [==============================] - 10s 105ms/step - loss: 10.7739 -
categorical_accuracy: 0.5562 - auc: 0.9109 - val_loss: 10.9294 -
val_categorical_accuracy: 0.4882 - val_auc: 0.8637
Epoch 12/150
93/93 [==============================] - 10s 105ms/step - loss: 10.4140 -
categorical_accuracy: 0.5637 - auc: 0.9138 - val_loss: 10.3381 -
val_categorical_accuracy: 0.5505 - val_auc: 0.8996
Epoch 13/150
93/93 [==============================] - 10s 111ms/step - loss: 10.0448 -
categorical_accuracy: 0.5777 - auc: 0.9196 - val_loss: 9.9985 -
val_categorical_accuracy: 0.5484 - val_auc: 0.9039
Epoch 14/150
93/93 [==============================] - 10s 107ms/step - loss: 9.7048 -
categorical_accuracy: 0.5788 - auc: 0.9217 - val_loss: 9.6860 -
val_categorical_accuracy: 0.5613 - val_auc: 0.8999
Epoch 15/150
93/93 [==============================] - 10s 104ms/step - loss: 9.3510 -
categorical_accuracy: 0.5885 - auc: 0.9280 - val_loss: 9.3050 -
val_categorical_accuracy: 0.5634 - val_auc: 0.9096
Epoch 16/150
93/93 [==============================] - 10s 106ms/step - loss: 9.0156 -
categorical_accuracy: 0.6025 - auc: 0.9308 - val_loss: 9.2879 -
val_categorical_accuracy: 0.4624 - val_auc: 0.8710
Epoch 17/150
93/93 [==============================] - 9s 101ms/step - loss: 8.7500 -
```

```
categorical_accuracy: 0.6025 - auc: 0.9264 - val_loss: 8.8560 -
val_categorical_accuracy: 0.5247 - val_auc: 0.8881
Epoch 18/150
93/93 [==============================] - 9s 102ms/step - loss: 8.4113 -
categorical_accuracy: 0.6062 - auc: 0.9323 - val_loss: 8.4067 -
val_categorical_accuracy: 0.5699 - val_auc: 0.9120
Epoch 19/150
93/93 [==============================] - 10s 107ms/step - loss: 8.0478 -
categorical_accuracy: 0.6434 - auc: 0.9426 - val_loss: 8.1703 -
val_categorical_accuracy: 0.5634 - val_auc: 0.9046
Epoch 20/150
93/93 [==============================] - 7s 70ms/step - loss: 7.8470 -
categorical_accuracy: 0.6154 - auc: 0.9318 - val_loss: 8.2339 -
val_categorical_accuracy: 0.3892 - val_auc: 0.8525
Epoch 21/150
93/93 [==============================] - 10s 104ms/step - loss: 7.4755 -
categorical_accuracy: 0.6487 - auc: 0.9455 - val_loss: 7.5938 -
val_categorical_accuracy: 0.5978 - val_auc: 0.9141
Epoch 22/150
93/93 [==============================] - 10s 103ms/step - loss: 7.1977 -
categorical_accuracy: 0.6606 - auc: 0.9473 - val_loss: 7.2940 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9191
Epoch 23/150
93/93 [==============================] - 10s 104ms/step - loss: 6.9486 -
categorical_accuracy: 0.6552 - auc: 0.9466 - val_loss: 7.2386 -
val_categorical_accuracy: 0.4882 - val_auc: 0.8904
Epoch 24/150
93/93 [==============================] - 10s 103ms/step - loss: 6.6549 -
categorical_accuracy: 0.6772 - auc: 0.9516 - val_loss: 6.7842 -
val_categorical_accuracy: 0.5763 - val_auc: 0.9203
Epoch 25/150
93/93 [==============================] - 9s 102ms/step - loss: 6.4173 -
categorical_accuracy: 0.6573 - auc: 0.9510 - val_loss: 6.5776 -
val_categorical_accuracy: 0.5699 - val_auc: 0.9160
Epoch 26/150
93/93 [==============================] - 10s 107ms/step - loss: 6.1591 -
categorical_accuracy: 0.6902 - auc: 0.9538 - val_loss: 6.3720 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9116
Epoch 27/150
93/93 [==============================] - 10s 104ms/step - loss: 5.9416 -
categorical_accuracy: 0.6724 - auc: 0.9531 - val_loss: 6.3580 -
val_categorical_accuracy: 0.5075 - val_auc: 0.8874
Epoch 28/150
93/93 [==============================] - 10s 108ms/step - loss: 5.7563 -
categorical_accuracy: 0.6509 - auc: 0.9489 - val_loss: 5.9449 -
val_categorical_accuracy: 0.5849 - val_auc: 0.9145
Epoch 29/150
93/93 [==============================] - 7s 70ms/step - loss: 5.4869 -
```

```
categorical_accuracy: 0.6762 - auc: 0.9552 - val_loss: 6.0333 -
val_categorical_accuracy: 0.5419 - val_auc: 0.8785
Epoch 30/150
93/93 [==============================] - 10s 110ms/step - loss: 5.2714 -
categorical_accuracy: 0.6821 - auc: 0.9559 - val_loss: 5.6144 -
val_categorical_accuracy: 0.5828 - val_auc: 0.9043
Epoch 31/150
93/93 [==============================] - 10s 103ms/step - loss: 5.0198 -
categorical_accuracy: 0.7133 - auc: 0.9607 - val_loss: 5.3068 -
val_categorical_accuracy: 0.5957 - val_auc: 0.9199
Epoch 32/150
93/93 [==============================] - 10s 108ms/step - loss: 4.8659 -
categorical_accuracy: 0.6934 - auc: 0.9566 - val_loss: 5.1500 -
val_categorical_accuracy: 0.5871 - val_auc: 0.9154
Epoch 33/150
93/93 [==============================] - 7s 70ms/step - loss: 4.6146 -
categorical_accuracy: 0.7084 - auc: 0.9627 - val_loss: 5.3304 -
val_categorical_accuracy: 0.5333 - val_auc: 0.8943
Epoch 34/150
93/93 [==============================] - 10s 108ms/step - loss: 4.4478 -
categorical_accuracy: 0.6950 - auc: 0.9611 - val_loss: 4.7334 -
val_categorical_accuracy: 0.5871 - val_auc: 0.9177
Epoch 35/150
93/93 [==============================] - 9s 102ms/step - loss: 4.2999 -
categorical_accuracy: 0.6880 - auc: 0.9580 - val_loss: 4.6108 -
val_categorical_accuracy: 0.5441 - val_auc: 0.9097
Epoch 36/150
93/93 [==============================] - 10s 109ms/step - loss: 4.0516 -
categorical_accuracy: 0.7117 - auc: 0.9656 - val_loss: 4.5220 -
val_categorical_accuracy: 0.5462 - val_auc: 0.8976
Epoch 37/150
93/93 [==============================] - 10s 106ms/step - loss: 3.9058 -
categorical_accuracy: 0.7144 - auc: 0.9630 - val_loss: 4.3589 -
val_categorical_accuracy: 0.5505 - val_auc: 0.8985
Epoch 38/150
93/93 [==============================] - 10s 104ms/step - loss: 3.7057 -
categorical_accuracy: 0.7133 - auc: 0.9663 - val_loss: 4.0514 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9180
Epoch 39/150
93/93 [==============================] - 10s 108ms/step - loss: 3.6281 -
categorical_accuracy: 0.7036 - auc: 0.9587 - val_loss: 3.8984 -
val_categorical_accuracy: 0.6022 - val_auc: 0.9177
Epoch 40/150
93/93 [==============================] - 10s 103ms/step - loss: 3.4427 -
categorical_accuracy: 0.7015 - auc: 0.9622 - val_loss: 3.8085 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9173
Epoch 41/150
93/93 [==============================] - 10s 105ms/step - loss: 3.2549 -
```

```
categorical_accuracy: 0.7127 - auc: 0.9670 - val_loss: 3.5729 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9262
Epoch 42/150
93/93 [==============================] - 7s 71ms/step - loss: 3.0855 -
categorical_accuracy: 0.7316 - auc: 0.9688 - val_loss: 3.7673 -
val_categorical_accuracy: 0.5527 - val_auc: 0.8814
Epoch 43/150
93/93 [==============================] - 10s 103ms/step - loss: 2.9756 -
categorical_accuracy: 0.7214 - auc: 0.9671 - val_loss: 3.5701 -
val_categorical_accuracy: 0.5699 - val_auc: 0.8985
Epoch 44/150
93/93 [==============================] - 10s 107ms/step - loss: 2.8946 -
categorical_accuracy: 0.7047 - auc: 0.9614 - val_loss: 3.2435 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9145
Epoch 45/150
93/93 [==============================] - 6s 70ms/step - loss: 2.7006 -
categorical_accuracy: 0.7294 - auc: 0.9686 - val_loss: 3.4729 -
val_categorical_accuracy: 0.5656 - val_auc: 0.8772
Epoch 46/150
93/93 [==============================] - 10s 107ms/step - loss: 2.5933 -
categorical_accuracy: 0.7197 - auc: 0.9670 - val_loss: 3.1083 -
val_categorical_accuracy: 0.5742 - val_auc: 0.9023
Epoch 47/150
93/93 [==============================] - 9s 102ms/step - loss: 2.4354 -
categorical_accuracy: 0.7434 - auc: 0.9706 - val_loss: 3.0138 -
val_categorical_accuracy: 0.5699 - val_auc: 0.9112
Epoch 48/150
93/93 [==============================] - 9s 101ms/step - loss: 2.4761 -
categorical_accuracy: 0.7025 - auc: 0.9564 - val_loss: 3.0118 -
val_categorical_accuracy: 0.5204 - val_auc: 0.8838
Epoch 49/150
93/93 [==============================] - 10s 108ms/step - loss: 2.2830 -
categorical_accuracy: 0.7230 - auc: 0.9657 - val_loss: 2.6836 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9224
Epoch 50/150
93/93 [==============================] - 9s 102ms/step - loss: 2.1717 -
categorical_accuracy: 0.7262 - auc: 0.9666 - val_loss: 2.6155 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9125
Epoch 51/150
93/93 [==============================] - 7s 75ms/step - loss: 2.0404 -
categorical_accuracy: 0.7289 - auc: 0.9694 - val_loss: 2.9417 -
val_categorical_accuracy: 0.5075 - val_auc: 0.8751
Epoch 52/150
93/93 [==============================] - 7s 70ms/step - loss: 1.9307 -
categorical_accuracy: 0.7439 - auc: 0.9708 - val_loss: 2.8948 -
val_categorical_accuracy: 0.5333 - val_auc: 0.8867
Epoch 53/150
93/93 [==============================] - 10s 109ms/step - loss: 1.8539 -
```

```
categorical_accuracy: 0.7208 - auc: 0.9696 - val_loss: 2.4401 -
val_categorical_accuracy: 0.5398 - val_auc: 0.9008
Epoch 54/150
93/93 [==============================] - 9s 101ms/step - loss: 1.8278 -
categorical_accuracy: 0.7208 - auc: 0.9649 - val_loss: 2.2426 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9136
Epoch 55/150
93/93 [==============================] - 6s 69ms/step - loss: 1.7311 -
categorical_accuracy: 0.7235 - auc: 0.9665 - val_loss: 2.3827 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9095
Epoch 56/150
93/93 [==============================] - 10s 109ms/step - loss: 1.6928 -
categorical_accuracy: 0.7025 - auc: 0.9629 - val_loss: 1.9848 -
val_categorical_accuracy: 0.6172 - val_auc: 0.9286
Epoch 57/150
93/93 [==============================] - 7s 70ms/step - loss: 1.5450 -
categorical_accuracy: 0.7332 - auc: 0.9696 - val_loss: 2.1319 -
val_categorical_accuracy: 0.5892 - val_auc: 0.8993
Epoch 58/150
93/93 [==============================] - 10s 110ms/step - loss: 1.4742 -
categorical_accuracy: 0.7472 - auc: 0.9702 - val_loss: 1.8543 -
val_categorical_accuracy: 0.6194 - val_auc: 0.9272
Epoch 59/150
93/93 [==============================] - 7s 70ms/step - loss: 1.4472 -
categorical_accuracy: 0.7278 - auc: 0.9674 - val_loss: 1.9622 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9021
Epoch 60/150
93/93 [==============================] - 7s 71ms/step - loss: 1.3739 -
categorical_accuracy: 0.7251 - auc: 0.9683 - val_loss: 1.9566 -
val_categorical_accuracy: 0.5871 - val_auc: 0.9046
Epoch 61/150
93/93 [==============================] - 7s 71ms/step - loss: 1.2958 -
categorical_accuracy: 0.7450 - auc: 0.9702 - val_loss: 3.2992 -
val_categorical_accuracy: 0.4194 - val_auc: 0.7932
Epoch 62/150
93/93 [==============================] - 7s 70ms/step - loss: 1.2176 -
categorical_accuracy: 0.7606 - auc: 0.9722 - val_loss: 1.9833 -
val_categorical_accuracy: 0.5075 - val_auc: 0.8952
Epoch 63/150
93/93 [==============================] - 7s 71ms/step - loss: 1.1819 -
categorical_accuracy: 0.7332 - auc: 0.9719 - val_loss: 1.8656 -
val_categorical_accuracy: 0.5785 - val_auc: 0.9068
Epoch 64/150
93/93 [==============================] - 10s 108ms/step - loss: 1.1391 -
categorical_accuracy: 0.7214 - auc: 0.9715 - val_loss: 1.6422 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9169
Epoch 65/150
93/93 [==============================] - 7s 70ms/step - loss: 1.0659 -
```

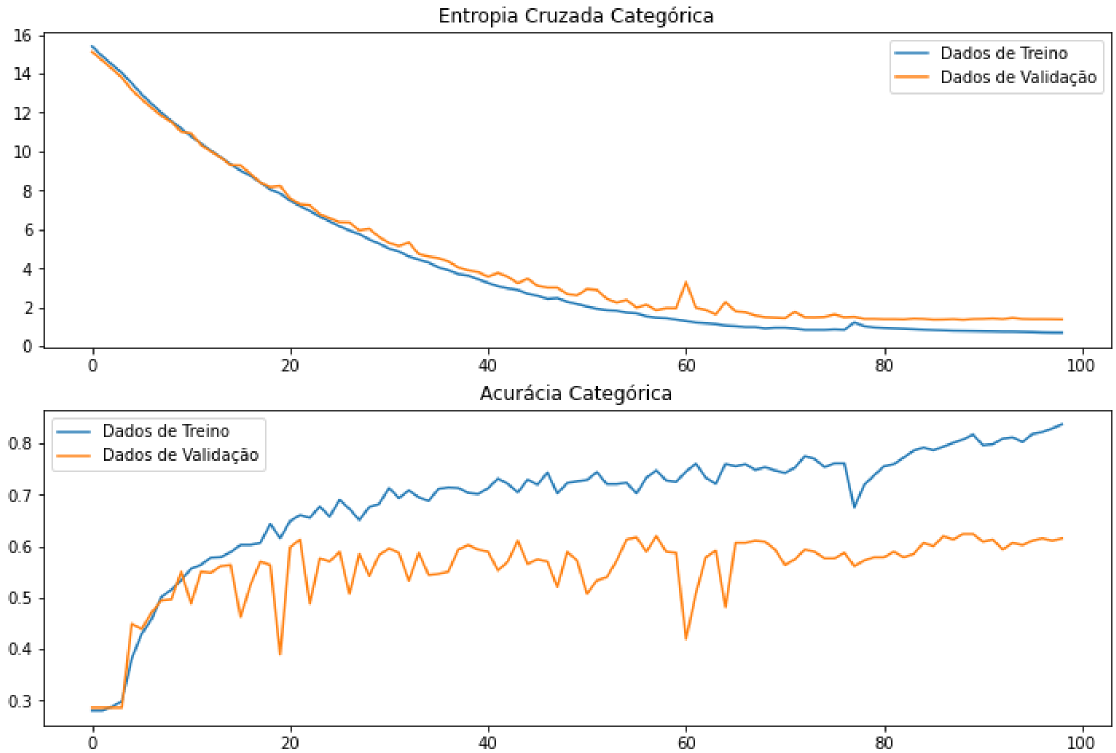
```
categorical_accuracy: 0.7601 - auc: 0.9746 - val_loss: 2.2692 -
val_categorical_accuracy: 0.4817 - val_auc: 0.8660
Epoch 66/150
93/93 [==============================] - 7s 71ms/step - loss: 1.0350 -
categorical_accuracy: 0.7552 - auc: 0.9739 - val_loss: 1.7988 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9103
Epoch 67/150
93/93 [==============================] - 7s 72ms/step - loss: 0.9909 -
categorical_accuracy: 0.7595 - auc: 0.9750 - val_loss: 1.7496 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9081
Epoch 68/150
93/93 [==============================] - 10s 103ms/step - loss: 0.9887 -
categorical_accuracy: 0.7483 - auc: 0.9732 - val_loss: 1.5853 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9169
Epoch 69/150
93/93 [==============================] - 10s 104ms/step - loss: 0.9194 -
categorical_accuracy: 0.7536 - auc: 0.9762 - val_loss: 1.4993 -
val_categorical_accuracy: 0.6086 - val_auc: 0.9180
Epoch 70/150
93/93 [==============================] - 10s 105ms/step - loss: 0.9560 -
categorical_accuracy: 0.7472 - auc: 0.9715 - val_loss: 1.4759 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9220
Epoch 71/150
93/93 [==============================] - 10s 105ms/step - loss: 0.9571 -
categorical_accuracy: 0.7423 - auc: 0.9689 - val_loss: 1.4499 -
val_categorical_accuracy: 0.5634 - val_auc: 0.9157
Epoch 72/150
93/93 [==============================] - 6s 70ms/step - loss: 0.9144 -
categorical_accuracy: 0.7526 - auc: 0.9732 - val_loss: 1.7696 -
val_categorical_accuracy: 0.5742 - val_auc: 0.8961
Epoch 73/150
93/93 [==============================] - 7s 70ms/step - loss: 0.8404 -
categorical_accuracy: 0.7751 - auc: 0.9779 - val_loss: 1.4973 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9200
Epoch 74/150
93/93 [==============================] - 7s 75ms/step - loss: 0.8402 -
categorical_accuracy: 0.7698 - auc: 0.9769 - val_loss: 1.4898 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9174
Epoch 75/150
93/93 [==============================] - 7s 71ms/step - loss: 0.8387 -
categorical_accuracy: 0.7536 - auc: 0.9769 - val_loss: 1.5119 -
val_categorical_accuracy: 0.5763 - val_auc: 0.9125
Epoch 76/150
93/93 [==============================] - 7s 75ms/step - loss: 0.8663 -
categorical_accuracy: 0.7612 - auc: 0.9745 - val_loss: 1.6384 -
val_categorical_accuracy: 0.5763 - val_auc: 0.9061
Epoch 77/150
93/93 [==============================] - 7s 70ms/step - loss: 0.8442 -
```

```
categorical_accuracy: 0.7612 - auc: 0.9762 - val_loss: 1.4942 -
val_categorical_accuracy: 0.5871 - val_auc: 0.9159
Epoch 78/150
93/93 [==============================] - ETA: 0s - loss: 1.2227 -
categorical_accuracy: 0.6751 - auc: 0.9383
Epoch 00078: ReduceLROnPlateau reducing learning rate to 0.005000000074505806.
93/93 [==============================] - 6s 70ms/step - loss: 1.2227 -
categorical_accuracy: 0.6751 - auc: 0.9383 - val_loss: 1.5187 -
val_categorical_accuracy: 0.5613 - val_auc: 0.9039
Epoch 79/150
93/93 [==============================] - 10s 102ms/step - loss: 1.0340 -
categorical_accuracy: 0.7192 - auc: 0.9633 - val_loss: 1.4108 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9146
Epoch 80/150
93/93 [==============================] - 7s 70ms/step - loss: 0.9708 -
categorical_accuracy: 0.7375 - auc: 0.9709 - val_loss: 1.4114 -
val_categorical_accuracy: 0.5785 - val_auc: 0.9161
Epoch 81/150
93/93 [==============================] - 10s 106ms/step - loss: 0.9426 -
categorical_accuracy: 0.7552 - auc: 0.9729 - val_loss: 1.3950 -
val_categorical_accuracy: 0.5785 - val_auc: 0.9183
Epoch 82/150
93/93 [==============================] - 6s 70ms/step - loss: 0.9210 -
categorical_accuracy: 0.7595 - auc: 0.9746 - val_loss: 1.3951 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9172
Epoch 83/150
93/93 [==============================] - 10s 103ms/step - loss: 0.9008 -
categorical_accuracy: 0.7725 - auc: 0.9761 - val_loss: 1.3839 -
val_categorical_accuracy: 0.5785 - val_auc: 0.9197
Epoch 84/150
93/93 [==============================] - 7s 75ms/step - loss: 0.8750 -
categorical_accuracy: 0.7859 - auc: 0.9778 - val_loss: 1.4262 -
val_categorical_accuracy: 0.5849 - val_auc: 0.9164
Epoch 85/150
93/93 [==============================] - 7s 70ms/step - loss: 0.8491 -
categorical_accuracy: 0.7918 - auc: 0.9794 - val_loss: 1.4105 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9170
Epoch 86/150
93/93 [==============================] - 10s 108ms/step - loss: 0.8315 -
categorical_accuracy: 0.7864 - auc: 0.9805 - val_loss: 1.3720 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9220
Epoch 87/150
93/93 [==============================] - 7s 72ms/step - loss: 0.8171 -
categorical_accuracy: 0.7934 - auc: 0.9814 - val_loss: 1.3777 -
val_categorical_accuracy: 0.6194 - val_auc: 0.9214
Epoch 88/150
93/93 [==============================] - 6s 70ms/step - loss: 0.7980 -
categorical_accuracy: 0.8010 - auc: 0.9825 - val_loss: 1.3974 -
```

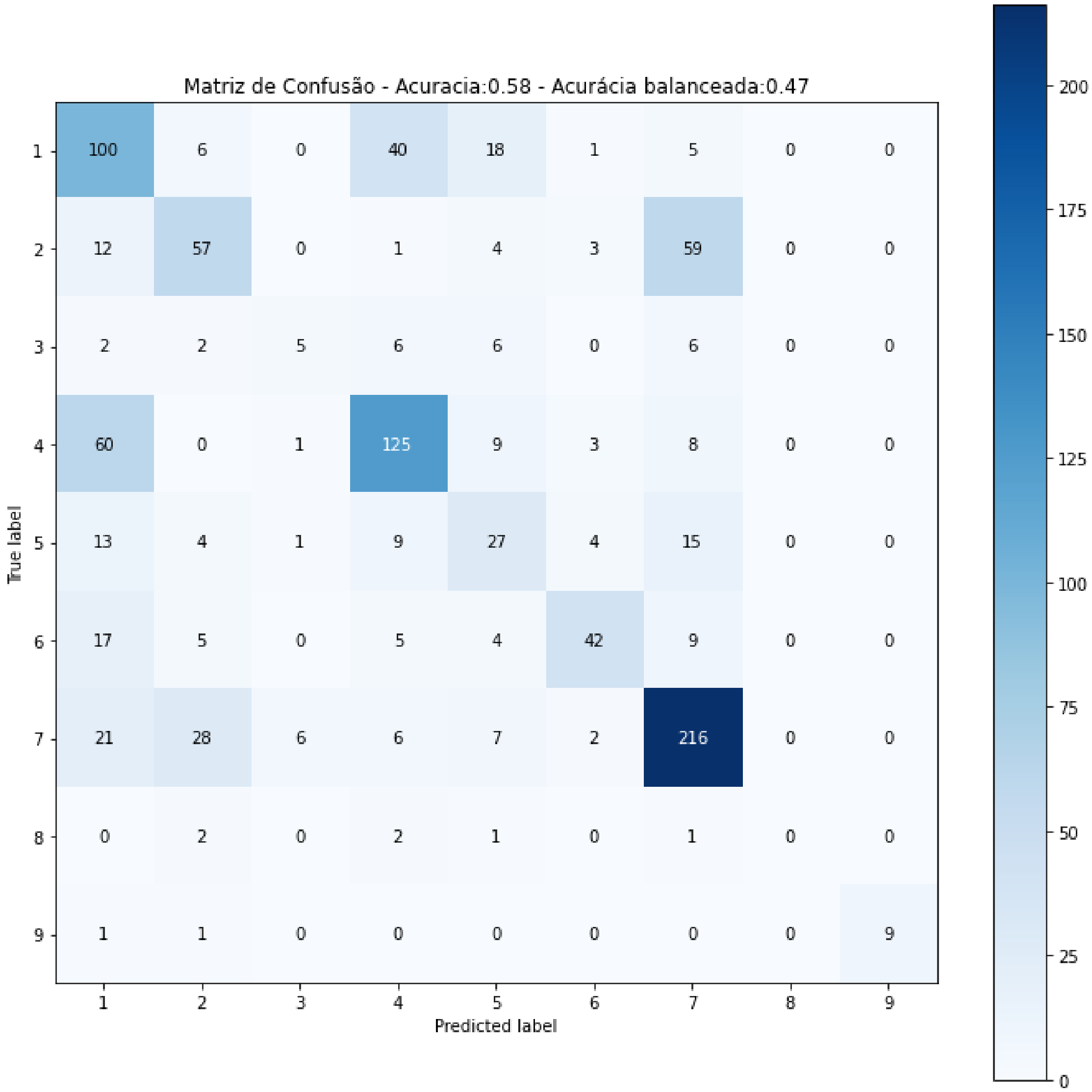
```
val_categorical_accuracy: 0.6129 - val_auc: 0.9191
Epoch 89/150
93/93 [==============================] - 10s 108ms/step - loss: 0.7908 -
categorical_accuracy: 0.8074 - auc: 0.9827 - val_loss: 1.3644 -
val_categorical_accuracy: 0.6237 - val_auc: 0.9230
Epoch 90/150
93/93 [==============================] - 7s 70ms/step - loss: 0.7828 -
categorical_accuracy: 0.8171 - auc: 0.9827 - val_loss: 1.4036 -
val_categorical_accuracy: 0.6237 - val_auc: 0.9191
Epoch 91/150
93/93 [==============================] - 7s 71ms/step - loss: 0.7739 -
categorical_accuracy: 0.7961 - auc: 0.9833 - val_loss: 1.4116 -
val_categorical_accuracy: 0.6086 - val_auc: 0.9189
Epoch 92/150
93/93 [==============================] - 7s 71ms/step - loss: 0.7661 -
categorical_accuracy: 0.7983 - auc: 0.9839 - val_loss: 1.4330 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9167
Epoch 93/150
93/93 [==============================] - 7s 72ms/step - loss: 0.7547 -
categorical_accuracy: 0.8090 - auc: 0.9844 - val_loss: 1.4008 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9218
Epoch 94/150
93/93 [==============================] - 7s 71ms/step - loss: 0.7536 -
categorical_accuracy: 0.8117 - auc: 0.9846 - val_loss: 1.4651 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9135
Epoch 95/150
93/93 [==============================] - 7s 73ms/step - loss: 0.7414 -
categorical_accuracy: 0.8026 - auc: 0.9853 - val_loss: 1.4060 -
val_categorical_accuracy: 0.6022 - val_auc: 0.9186
Epoch 96/150
93/93 [==============================] - ETA: 0s - loss: 0.7298 -
categorical_accuracy: 0.8182 - auc: 0.9861
Epoch 00096: ReduceLROnPlateau reducing learning rate to 0.0004999999888241291.
93/93 [==============================] - 6s 70ms/step - loss: 0.7298 -
categorical_accuracy: 0.8182 - auc: 0.9861 - val_loss: 1.3957 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9218
Epoch 97/150
93/93 [==============================] - 7s 70ms/step - loss: 0.7115 -
categorical_accuracy: 0.8219 - auc: 0.9872 - val_loss: 1.3973 -
val_categorical_accuracy: 0.6151 - val_auc: 0.9221
Epoch 98/150
93/93 [==============================] - 7s 71ms/step - loss: 0.7035 -
categorical_accuracy: 0.8289 - auc: 0.9879 - val_loss: 1.3920 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9220
Epoch 99/150
93/93 [==============================] - 7s 75ms/step - loss: 0.7013 -
categorical_accuracy: 0.8370 - auc: 0.9878 - val_loss: 1.3839 -
val_categorical_accuracy: 0.6151 - val_auc: 0.9233
```

```
Epoch 00099: early stopping
```

[111]:
```
modelo3_1_history = json.load(open("./modelos/history_modelo3_1.json",'r'))
plot_treinamento(modelo3_1_history)
```



[51]:
```
plot_matriz_confusao(modelo3_1,X_test_novo,Y_test_novo)
```



Houve uma aparente melhora em relação ao modelo 3 original

2 - incluindo regularização no modelo e reduzindo ainda mais a quantidade de neuronios na camada de entrada

[114]:
```
reset_keras()

camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name = "Camada_Entrada")

primeira_camada_oculta = Dense(300,activation = 'relu',kernel_initializer = 'uniform',
                                kernel_regularizer = regularizers.l1_l2(l1=1e-5, l2=1e-4),
```

```
                                bias_regularizer=regularizers.l2(1e-4),
                                activity_regularizer=regularizers.l2(1e-5),
                                name = "Camada_Oculta_1")(camada_entrada)

segunda_camada_oculta = Dense(50,activation = 'relu',kernel_initializer =␣
 ↪'uniform',
                               kernel_regularizer = regularizers.l1_l2(l1=1e-5,␣
 ↪l2=1e-4),
                               bias_regularizer=regularizers.l2(1e-4),
                               activity_regularizer=regularizers.l2(1e-5),
                               name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer =␣
 ↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo3_2 = Model(inputs = [camada_entrada], outputs = [camada_saida])

modelo3_2.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])

modelo3_2 =␣
 ↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
               saving_checkpoint_path="./modelos/", nome_modelo="modelo3_2",␣
 ↪modelo= modelo3_2)
```

```
3217
Epoch 1/150
93/93 [==============================] - 8s 89ms/step - loss: 10.0001 -
categorical_accuracy: 0.2813 - auc: 0.7386 - val_loss: 9.8026 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7532
Epoch 2/150
93/93 [==============================] - 6s 68ms/step - loss: 9.6720 -
categorical_accuracy: 0.2873 - auc: 0.7563 - val_loss: 9.5433 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7520
Epoch 3/150
93/93 [==============================] - 6s 65ms/step - loss: 9.4168 -
categorical_accuracy: 0.2873 - auc: 0.7558 - val_loss: 9.2881 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7612
Epoch 4/150
93/93 [==============================] - 6s 66ms/step - loss: 9.1628 -
categorical_accuracy: 0.2835 - auc: 0.7581 - val_loss: 9.0418 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7584
Epoch 5/150
93/93 [==============================] - 6s 68ms/step - loss: 8.9174 -
categorical_accuracy: 0.2873 - auc: 0.7581 - val_loss: 8.7937 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7701
```

```
Epoch 6/150
93/93 [==============================] - 6s 65ms/step - loss: 8.6672 -
categorical_accuracy: 0.2932 - auc: 0.7634 - val_loss: 8.5198 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7829
Epoch 7/150
93/93 [==============================] - 6s 68ms/step - loss: 8.3325 -
categorical_accuracy: 0.3658 - auc: 0.8017 - val_loss: 8.0721 -
val_categorical_accuracy: 0.4516 - val_auc: 0.8514
Epoch 8/150
93/93 [==============================] - 6s 66ms/step - loss: 7.9206 -
categorical_accuracy: 0.4411 - auc: 0.8500 - val_loss: 7.7236 -
val_categorical_accuracy: 0.4774 - val_auc: 0.8662
Epoch 9/150
93/93 [==============================] - 6s 66ms/step - loss: 7.6417 -
categorical_accuracy: 0.4524 - auc: 0.8599 - val_loss: 7.4862 -
val_categorical_accuracy: 0.4667 - val_auc: 0.8726
Epoch 10/150
93/93 [==============================] - 6s 66ms/step - loss: 7.3808 -
categorical_accuracy: 0.4642 - auc: 0.8699 - val_loss: 7.4317 -
val_categorical_accuracy: 0.4151 - val_auc: 0.8381
Epoch 11/150
93/93 [==============================] - 6s 67ms/step - loss: 7.1570 -
categorical_accuracy: 0.4691 - auc: 0.8731 - val_loss: 7.0419 -
val_categorical_accuracy: 0.4624 - val_auc: 0.8767
Epoch 12/150
93/93 [==============================] - 6s 66ms/step - loss: 6.9346 -
categorical_accuracy: 0.4669 - auc: 0.8776 - val_loss: 6.7900 -
val_categorical_accuracy: 0.4774 - val_auc: 0.8847
Epoch 13/150
93/93 [==============================] - 6s 66ms/step - loss: 6.6561 -
categorical_accuracy: 0.4987 - auc: 0.8905 - val_loss: 6.5315 -
val_categorical_accuracy: 0.5183 - val_auc: 0.8955
Epoch 14/150
93/93 [==============================] - 6s 66ms/step - loss: 6.4039 -
categorical_accuracy: 0.5304 - auc: 0.9003 - val_loss: 6.3009 -
val_categorical_accuracy: 0.5355 - val_auc: 0.9022
Epoch 15/150
93/93 [==============================] - 6s 65ms/step - loss: 6.1662 -
categorical_accuracy: 0.5471 - auc: 0.9079 - val_loss: 6.2124 -
val_categorical_accuracy: 0.5269 - val_auc: 0.8885
Epoch 16/150
93/93 [==============================] - 6s 66ms/step - loss: 5.9561 -
categorical_accuracy: 0.5503 - auc: 0.9122 - val_loss: 5.9011 -
val_categorical_accuracy: 0.5634 - val_auc: 0.9089
Epoch 17/150
93/93 [==============================] - 5s 49ms/step - loss: 5.7765 -
categorical_accuracy: 0.5578 - auc: 0.9126 - val_loss: 5.9293 -
val_categorical_accuracy: 0.5118 - val_auc: 0.8782
```

```
Epoch 18/150
93/93 [==============================] - 6s 64ms/step - loss: 5.5501 -
categorical_accuracy: 0.5718 - auc: 0.9206 - val_loss: 5.5239 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9125
Epoch 19/150
93/93 [==============================] - 6s 68ms/step - loss: 5.3509 -
categorical_accuracy: 0.5729 - auc: 0.9245 - val_loss: 5.3293 -
val_categorical_accuracy: 0.6086 - val_auc: 0.9161
Epoch 20/150
93/93 [==============================] - 6s 65ms/step - loss: 5.1808 -
categorical_accuracy: 0.5842 - auc: 0.9257 - val_loss: 5.1814 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9122
Epoch 21/150
93/93 [==============================] - 4s 48ms/step - loss: 4.9657 -
categorical_accuracy: 0.5998 - auc: 0.9331 - val_loss: 5.8445 -
val_categorical_accuracy: 0.4989 - val_auc: 0.8256
Epoch 22/150
93/93 [==============================] - 6s 65ms/step - loss: 4.8243 -
categorical_accuracy: 0.6116 - auc: 0.9310 - val_loss: 4.8443 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9161
Epoch 23/150
93/93 [==============================] - 4s 48ms/step - loss: 4.6227 -
categorical_accuracy: 0.6267 - auc: 0.9376 - val_loss: 5.3769 -
val_categorical_accuracy: 0.3892 - val_auc: 0.8252
Epoch 24/150
93/93 [==============================] - 6s 65ms/step - loss: 4.4738 -
categorical_accuracy: 0.6353 - auc: 0.9380 - val_loss: 4.6794 -
val_categorical_accuracy: 0.5742 - val_auc: 0.8983
Epoch 25/150
93/93 [==============================] - 6s 66ms/step - loss: 4.3266 -
categorical_accuracy: 0.6267 - auc: 0.9384 - val_loss: 4.6735 -
val_categorical_accuracy: 0.5763 - val_auc: 0.8865
Epoch 26/150
93/93 [==============================] - 6s 67ms/step - loss: 4.2122 -
categorical_accuracy: 0.6347 - auc: 0.9352 - val_loss: 4.2760 -
val_categorical_accuracy: 0.6022 - val_auc: 0.9153
Epoch 27/150
93/93 [==============================] - 6s 67ms/step - loss: 4.0181 -
categorical_accuracy: 0.6520 - auc: 0.9426 - val_loss: 4.1482 -
val_categorical_accuracy: 0.6043 - val_auc: 0.9171
Epoch 28/150
93/93 [==============================] - 6s 65ms/step - loss: 3.9020 -
categorical_accuracy: 0.6487 - auc: 0.9406 - val_loss: 4.0807 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9142
Epoch 29/150
93/93 [==============================] - 6s 68ms/step - loss: 3.8132 -
categorical_accuracy: 0.6342 - auc: 0.9368 - val_loss: 3.8775 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9192
```

```
Epoch 30/150
93/93 [==============================] - 4s 48ms/step - loss: 3.6059 -
categorical_accuracy: 0.6552 - auc: 0.9468 - val_loss: 4.0845 -
val_categorical_accuracy: 0.5355 - val_auc: 0.8811
Epoch 31/150
93/93 [==============================] - 6s 65ms/step - loss: 3.4788 -
categorical_accuracy: 0.6493 - auc: 0.9475 - val_loss: 3.8285 -
val_categorical_accuracy: 0.5871 - val_auc: 0.8941
Epoch 32/150
93/93 [==============================] - 6s 68ms/step - loss: 3.3240 -
categorical_accuracy: 0.6762 - auc: 0.9510 - val_loss: 3.5660 -
val_categorical_accuracy: 0.5763 - val_auc: 0.9126
Epoch 33/150
93/93 [==============================] - 5s 50ms/step - loss: 3.1966 -
categorical_accuracy: 0.6832 - auc: 0.9537 - val_loss: 3.8851 -
val_categorical_accuracy: 0.4151 - val_auc: 0.8586
Epoch 34/150
93/93 [==============================] - 6s 67ms/step - loss: 3.0999 -
categorical_accuracy: 0.6821 - auc: 0.9515 - val_loss: 3.4622 -
val_categorical_accuracy: 0.5312 - val_auc: 0.8992
Epoch 35/150
93/93 [==============================] - 6s 66ms/step - loss: 3.2065 -
categorical_accuracy: 0.6143 - auc: 0.9256 - val_loss: 3.4364 -
val_categorical_accuracy: 0.5548 - val_auc: 0.8877
Epoch 36/150
93/93 [==============================] - 6s 67ms/step - loss: 2.9541 -
categorical_accuracy: 0.6627 - auc: 0.9454 - val_loss: 3.1092 -
val_categorical_accuracy: 0.5957 - val_auc: 0.9166
Epoch 37/150
93/93 [==============================] - 4s 48ms/step - loss: 2.8165 -
categorical_accuracy: 0.6724 - auc: 0.9501 - val_loss: 3.2605 -
val_categorical_accuracy: 0.5290 - val_auc: 0.8946
Epoch 38/150
93/93 [==============================] - 4s 47ms/step - loss: 2.7157 -
categorical_accuracy: 0.6805 - auc: 0.9494 - val_loss: 3.2077 -
val_categorical_accuracy: 0.4903 - val_auc: 0.8786
Epoch 39/150
93/93 [==============================] - 6s 65ms/step - loss: 2.6859 -
categorical_accuracy: 0.6557 - auc: 0.9431 - val_loss: 2.9589 -
val_categorical_accuracy: 0.5548 - val_auc: 0.8976
Epoch 40/150
93/93 [==============================] - 4s 48ms/step - loss: 2.5458 -
categorical_accuracy: 0.6579 - auc: 0.9493 - val_loss: 3.0660 -
val_categorical_accuracy: 0.4753 - val_auc: 0.8785
Epoch 41/150
93/93 [==============================] - 6s 65ms/step - loss: 2.4130 -
categorical_accuracy: 0.6869 - auc: 0.9535 - val_loss: 2.8253 -
val_categorical_accuracy: 0.5720 - val_auc: 0.8993
```

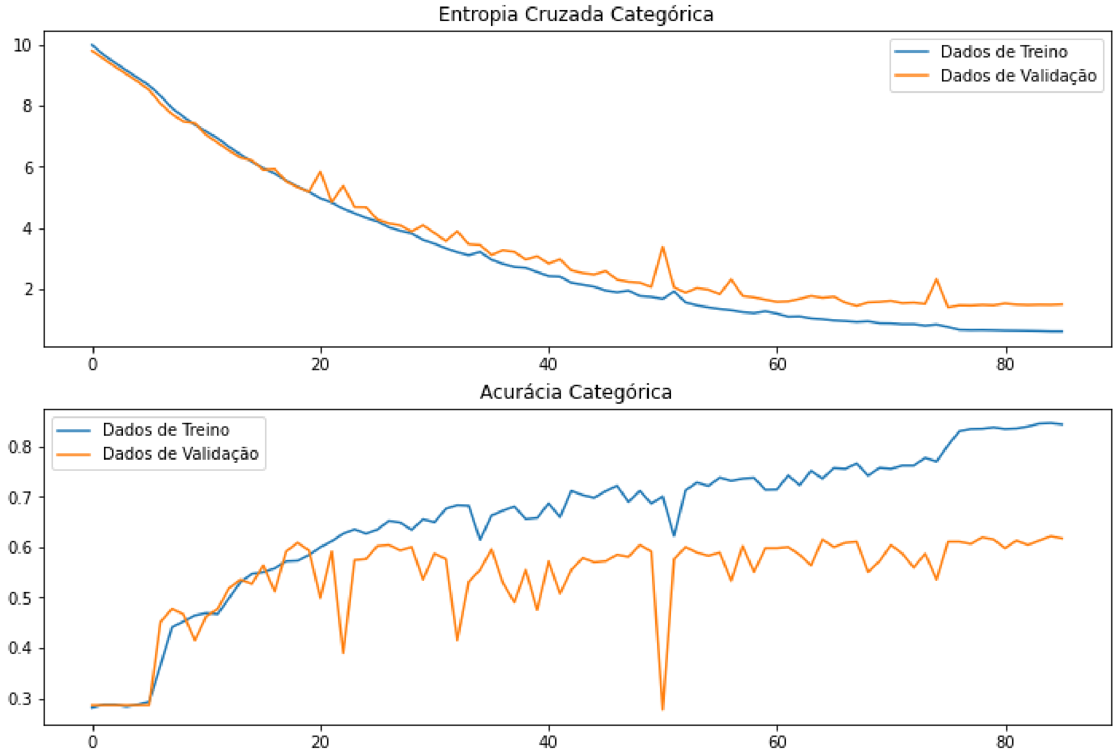
```
Epoch 42/150
93/93 [==============================] - 4s 47ms/step - loss: 2.3961 -
categorical_accuracy: 0.6595 - auc: 0.9463 - val_loss: 2.9684 -
val_categorical_accuracy: 0.5075 - val_auc: 0.8721
Epoch 43/150
93/93 [==============================] - 6s 65ms/step - loss: 2.1975 -
categorical_accuracy: 0.7117 - auc: 0.9603 - val_loss: 2.6029 -
val_categorical_accuracy: 0.5548 - val_auc: 0.9052
Epoch 44/150
93/93 [==============================] - 6s 68ms/step - loss: 2.1299 -
categorical_accuracy: 0.7031 - auc: 0.9588 - val_loss: 2.5110 -
val_categorical_accuracy: 0.5785 - val_auc: 0.9106
Epoch 45/150
93/93 [==============================] - 6s 65ms/step - loss: 2.0659 -
categorical_accuracy: 0.6977 - auc: 0.9575 - val_loss: 2.4634 -
val_categorical_accuracy: 0.5699 - val_auc: 0.9023
Epoch 46/150
93/93 [==============================] - 4s 48ms/step - loss: 1.9407 -
categorical_accuracy: 0.7111 - auc: 0.9633 - val_loss: 2.5796 -
val_categorical_accuracy: 0.5720 - val_auc: 0.8925
Epoch 47/150
93/93 [==============================] - 6s 66ms/step - loss: 1.8817 -
categorical_accuracy: 0.7219 - auc: 0.9618 - val_loss: 2.2985 -
val_categorical_accuracy: 0.5849 - val_auc: 0.9082
Epoch 48/150
93/93 [==============================] - 6s 67ms/step - loss: 1.9385 -
categorical_accuracy: 0.6896 - auc: 0.9494 - val_loss: 2.2261 -
val_categorical_accuracy: 0.5806 - val_auc: 0.9070
Epoch 49/150
93/93 [==============================] - 6s 68ms/step - loss: 1.7607 -
categorical_accuracy: 0.7117 - auc: 0.9619 - val_loss: 2.1971 -
val_categorical_accuracy: 0.6043 - val_auc: 0.9122
Epoch 50/150
93/93 [==============================] - 6s 66ms/step - loss: 1.7300 -
categorical_accuracy: 0.6864 - auc: 0.9583 - val_loss: 2.0585 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9133
Epoch 51/150
93/93 [==============================] - 4s 48ms/step - loss: 1.6645 -
categorical_accuracy: 0.7004 - auc: 0.9586 - val_loss: 3.3778 -
val_categorical_accuracy: 0.2774 - val_auc: 0.6809
Epoch 52/150
93/93 [==============================] - 6s 66ms/step - loss: 1.9092 -
categorical_accuracy: 0.6224 - auc: 0.9239 - val_loss: 2.0456 -
val_categorical_accuracy: 0.5763 - val_auc: 0.9003
Epoch 53/150
93/93 [==============================] - 6s 66ms/step - loss: 1.5500 -
categorical_accuracy: 0.7127 - auc: 0.9605 - val_loss: 1.8685 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9194
```

```
Epoch 54/150
93/93 [==============================] - 4s 47ms/step - loss: 1.4557 -
categorical_accuracy: 0.7283 - auc: 0.9653 - val_loss: 2.0245 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9045
Epoch 55/150
93/93 [==============================] - 4s 47ms/step - loss: 1.3881 -
categorical_accuracy: 0.7214 - auc: 0.9665 - val_loss: 1.9666 -
val_categorical_accuracy: 0.5828 - val_auc: 0.9092
Epoch 56/150
93/93 [==============================] - 6s 66ms/step - loss: 1.3356 -
categorical_accuracy: 0.7380 - auc: 0.9670 - val_loss: 1.8193 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9130
Epoch 57/150
93/93 [==============================] - 4s 47ms/step - loss: 1.2952 -
categorical_accuracy: 0.7316 - auc: 0.9672 - val_loss: 2.3050 -
val_categorical_accuracy: 0.5333 - val_auc: 0.8624
Epoch 58/150
93/93 [==============================] - 6s 65ms/step - loss: 1.2318 -
categorical_accuracy: 0.7359 - auc: 0.9687 - val_loss: 1.7612 -
val_categorical_accuracy: 0.6022 - val_auc: 0.9055
Epoch 59/150
93/93 [==============================] - 6s 65ms/step - loss: 1.1999 -
categorical_accuracy: 0.7375 - auc: 0.9686 - val_loss: 1.7139 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9052
Epoch 60/150
93/93 [==============================] - 6s 67ms/step - loss: 1.2616 -
categorical_accuracy: 0.7138 - auc: 0.9591 - val_loss: 1.6380 -
val_categorical_accuracy: 0.5978 - val_auc: 0.9164
Epoch 61/150
93/93 [==============================] - 6s 69ms/step - loss: 1.1844 -
categorical_accuracy: 0.7144 - auc: 0.9635 - val_loss: 1.5684 -
val_categorical_accuracy: 0.5978 - val_auc: 0.9192
Epoch 62/150
93/93 [==============================] - 5s 49ms/step - loss: 1.0818 -
categorical_accuracy: 0.7423 - auc: 0.9705 - val_loss: 1.5828 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9196
Epoch 63/150
93/93 [==============================] - 4s 47ms/step - loss: 1.0915 -
categorical_accuracy: 0.7230 - auc: 0.9672 - val_loss: 1.6641 -
val_categorical_accuracy: 0.5849 - val_auc: 0.9031
Epoch 64/150
93/93 [==============================] - 4s 47ms/step - loss: 1.0246 -
categorical_accuracy: 0.7515 - auc: 0.9707 - val_loss: 1.7621 -
val_categorical_accuracy: 0.5634 - val_auc: 0.8978
Epoch 65/150
93/93 [==============================] - 4s 47ms/step - loss: 0.9983 -
categorical_accuracy: 0.7359 - auc: 0.9708 - val_loss: 1.6998 -
val_categorical_accuracy: 0.6151 - val_auc: 0.9021
```

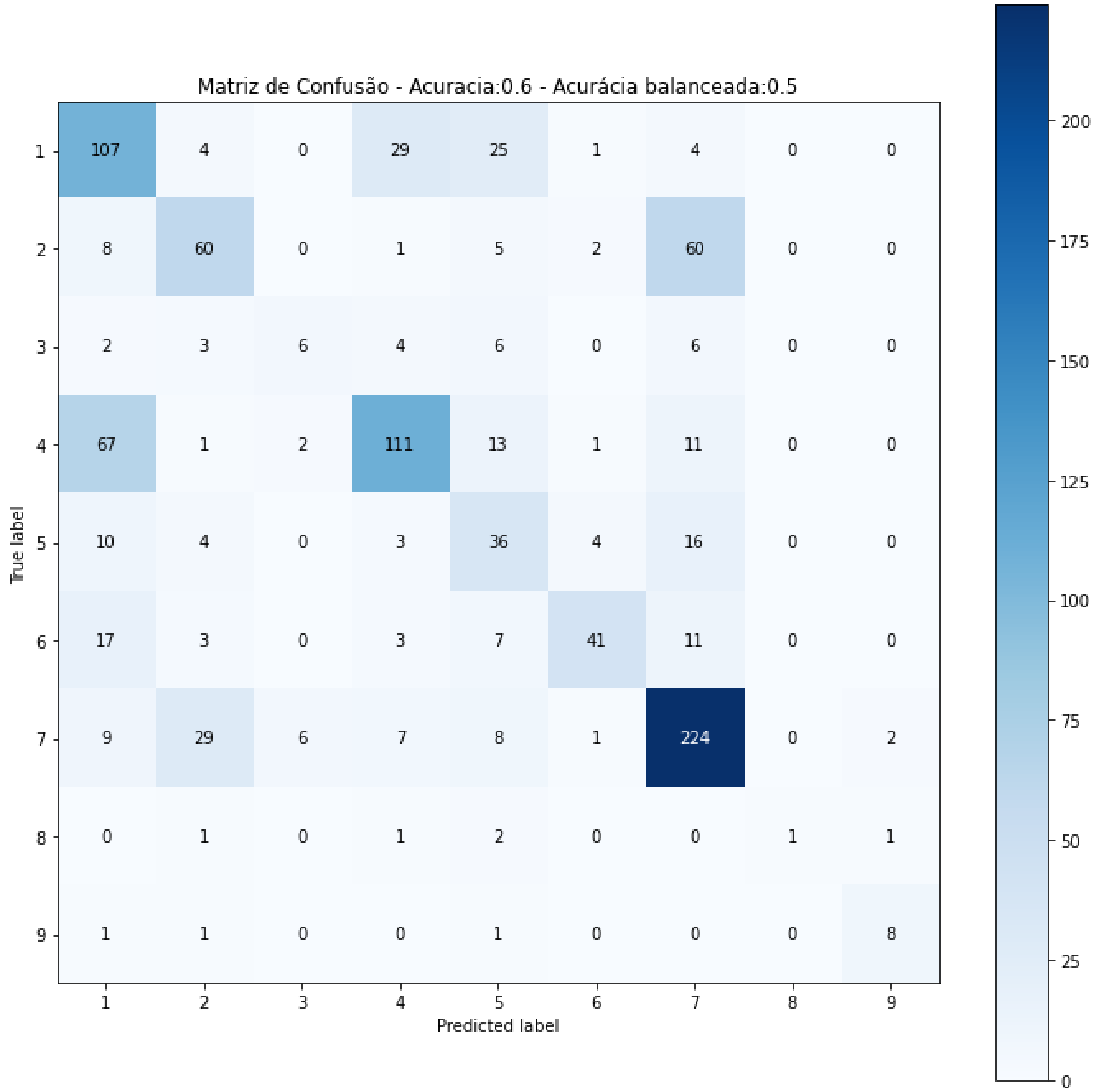
```
Epoch 66/150
93/93 [==============================] - 4s 48ms/step - loss: 0.9574 -
categorical_accuracy: 0.7569 - auc: 0.9723 - val_loss: 1.7376 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9039
Epoch 67/150
93/93 [==============================] - 6s 66ms/step - loss: 0.9378 -
categorical_accuracy: 0.7552 - auc: 0.9722 - val_loss: 1.5425 -
val_categorical_accuracy: 0.6086 - val_auc: 0.9129
Epoch 68/150
93/93 [==============================] - 6s 65ms/step - loss: 0.9090 -
categorical_accuracy: 0.7660 - auc: 0.9738 - val_loss: 1.4392 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9214
Epoch 69/150
93/93 [==============================] - 4s 48ms/step - loss: 0.9300 -
categorical_accuracy: 0.7418 - auc: 0.9708 - val_loss: 1.5498 -
val_categorical_accuracy: 0.5505 - val_auc: 0.9052
Epoch 70/150
93/93 [==============================] - 4s 47ms/step - loss: 0.8661 -
categorical_accuracy: 0.7574 - auc: 0.9744 - val_loss: 1.5674 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9048
Epoch 71/150
93/93 [==============================] - 4s 47ms/step - loss: 0.8608 -
categorical_accuracy: 0.7552 - auc: 0.9742 - val_loss: 1.6007 -
val_categorical_accuracy: 0.6043 - val_auc: 0.9079
Epoch 72/150
93/93 [==============================] - 4s 47ms/step - loss: 0.8342 -
categorical_accuracy: 0.7617 - auc: 0.9761 - val_loss: 1.5267 -
val_categorical_accuracy: 0.5871 - val_auc: 0.9055
Epoch 73/150
93/93 [==============================] - 4s 47ms/step - loss: 0.8344 -
categorical_accuracy: 0.7617 - auc: 0.9750 - val_loss: 1.5445 -
val_categorical_accuracy: 0.5591 - val_auc: 0.8983
Epoch 74/150
93/93 [==============================] - 4s 47ms/step - loss: 0.7862 -
categorical_accuracy: 0.7773 - auc: 0.9791 - val_loss: 1.5017 -
val_categorical_accuracy: 0.5871 - val_auc: 0.9143
Epoch 75/150
93/93 [==============================] - ETA: 0s - loss: 0.8227 -
categorical_accuracy: 0.7698 - auc: 0.9758
Epoch 00075: ReduceLROnPlateau reducing learning rate to 0.005000000074505806.
93/93 [==============================] - 4s 47ms/step - loss: 0.8227 -
categorical_accuracy: 0.7698 - auc: 0.9758 - val_loss: 2.3152 -
val_categorical_accuracy: 0.5355 - val_auc: 0.8586
Epoch 76/150
93/93 [==============================] - 6s 68ms/step - loss: 0.7429 -
categorical_accuracy: 0.8020 - auc: 0.9811 - val_loss: 1.3887 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9227
Epoch 77/150
```

```
93/93 [==============================] - 4s 47ms/step - loss: 0.6519 -
categorical_accuracy: 0.8300 - auc: 0.9881 - val_loss: 1.4570 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9179
Epoch 78/150
93/93 [==============================] - 4s 47ms/step - loss: 0.6421 -
categorical_accuracy: 0.8343 - auc: 0.9886 - val_loss: 1.4495 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9214
Epoch 79/150
93/93 [==============================] - 4s 47ms/step - loss: 0.6448 -
categorical_accuracy: 0.8349 - auc: 0.9880 - val_loss: 1.4706 -
val_categorical_accuracy: 0.6194 - val_auc: 0.9127
Epoch 80/150
93/93 [==============================] - 4s 47ms/step - loss: 0.6378 -
categorical_accuracy: 0.8375 - auc: 0.9883 - val_loss: 1.4568 -
val_categorical_accuracy: 0.6151 - val_auc: 0.9181
Epoch 81/150
93/93 [==============================] - 4s 46ms/step - loss: 0.6285 -
categorical_accuracy: 0.8343 - auc: 0.9891 - val_loss: 1.5179 -
val_categorical_accuracy: 0.5978 - val_auc: 0.9159
Epoch 82/150
93/93 [==============================] - 4s 47ms/step - loss: 0.6247 -
categorical_accuracy: 0.8354 - auc: 0.9893 - val_loss: 1.4797 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9192
Epoch 83/150
93/93 [==============================] - ETA: 0s - loss: 0.6189 -
categorical_accuracy: 0.8392 - auc: 0.9896
Epoch 00083: ReduceLROnPlateau reducing learning rate to 0.0004999999888241291.
93/93 [==============================] - 4s 47ms/step - loss: 0.6189 -
categorical_accuracy: 0.8392 - auc: 0.9896 - val_loss: 1.4670 -
val_categorical_accuracy: 0.6043 - val_auc: 0.9188
Epoch 84/150
93/93 [==============================] - 4s 47ms/step - loss: 0.6118 -
categorical_accuracy: 0.8451 - auc: 0.9899 - val_loss: 1.4756 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9192
Epoch 85/150
93/93 [==============================] - 5s 52ms/step - loss: 0.6010 -
categorical_accuracy: 0.8462 - auc: 0.9906 - val_loss: 1.4725 -
val_categorical_accuracy: 0.6215 - val_auc: 0.9192
Epoch 86/150
93/93 [==============================] - 4s 48ms/step - loss: 0.5999 -
categorical_accuracy: 0.8435 - auc: 0.9905 - val_loss: 1.4874 -
val_categorical_accuracy: 0.6172 - val_auc: 0.9174
Epoch 00086: early stopping
```

[115]:
```
modelo3_2_history = json.load(open("./modelos/history_modelo3_2.json",'r'))
plot_treinamento(modelo3_2_history)
```



[50]:
```
plot_matriz_confusao(modelo3_2,X_test_novo,Y_test_novo)
```



Houve melhora em relação ao modelo 3_1

3 - incluindo regularização no modelo e reduzindo ainda mais a quantidade de neuronios na camada de entrada

[59]:
```
reset_keras()

camada_entrada = Input(shape = (X_novo.shape[1],), sparse=True,name =␣
↪"Camada_Entrada")

primeira_camada_oculta = Dense(200,activation = 'relu',kernel_initializer =␣
↪'uniform',
                                kernel_regularizer = regularizers.l1_l2(l1=1e-5,␣
↪l2=1e-4),
```

```
                                bias_regularizer=regularizers.l2(1e-4),
                                activity_regularizer=regularizers.l2(1e-5),
                                name = "Camada_Oculta_1")(camada_entrada)

segunda_camada_oculta = Dense(50,activation = 'relu',kernel_initializer =␣
 ↪'uniform',
                               kernel_regularizer = regularizers.l1_l2(l1=1e-5,␣
 ↪l2=1e-4),
                               bias_regularizer=regularizers.l2(1e-4),
                               activity_regularizer=regularizers.l2(1e-5),
                               name = "Camada_Oculta_2")(primeira_camada_oculta)

camada_saida = Dense(9,activation = 'softmax',kernel_initializer =␣
 ↪'uniform',name = "Camada_de_Saida")(segunda_camada_oculta)

modelo3_3 = Model(inputs = [camada_entrada], outputs = [camada_saida])

modelo3_3.compile(loss = 'categorical_crossentropy',
                optimizer = SGD(lr = 0.05, momentum = 0.9, nesterov = True),
                metrics = ['categorical_accuracy',AUC()])

modelo3_3 =␣
 ↪start_training(X_train=X_train_novo,X_valid=X_valid_novo,Y_train=Y_train_novo,Y_valid=Y_val
               saving_checkpoint_path="./modelos/", nome_modelo="modelo3_3",␣
 ↪modelo= modelo3_3)
```

```
22803
Epoch 1/150
93/93 [==============================] - 6s 61ms/step - loss: 7.3022 -
categorical_accuracy: 0.2792 - auc: 0.7401 - val_loss: 7.1532 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7532
Epoch 2/150
93/93 [==============================] - 4s 43ms/step - loss: 7.0633 -
categorical_accuracy: 0.2873 - auc: 0.7548 - val_loss: 6.9707 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7618
Epoch 3/150
93/93 [==============================] - 4s 41ms/step - loss: 6.8900 -
categorical_accuracy: 0.2873 - auc: 0.7556 - val_loss: 6.8038 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7619
Epoch 4/150
93/93 [==============================] - 4s 41ms/step - loss: 6.7238 -
categorical_accuracy: 0.2873 - auc: 0.7546 - val_loss: 6.6376 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7592
Epoch 5/150
93/93 [==============================] - 4s 42ms/step - loss: 6.5589 -
categorical_accuracy: 0.2781 - auc: 0.7565 - val_loss: 6.4735 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7630
```

```
Epoch 6/150
93/93 [==============================] - 4s 42ms/step - loss: 6.3969 -
categorical_accuracy: 0.2873 - auc: 0.7579 - val_loss: 6.3175 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7591
Epoch 7/150
93/93 [==============================] - 4s 41ms/step - loss: 6.2437 -
categorical_accuracy: 0.2873 - auc: 0.7565 - val_loss: 6.1601 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7652
Epoch 8/150
93/93 [==============================] - 4s 40ms/step - loss: 6.0825 -
categorical_accuracy: 0.2932 - auc: 0.7619 - val_loss: 5.9917 -
val_categorical_accuracy: 0.2860 - val_auc: 0.7806
Epoch 9/150
93/93 [==============================] - 4s 43ms/step - loss: 5.8758 -
categorical_accuracy: 0.3378 - auc: 0.7848 - val_loss: 5.6972 -
val_categorical_accuracy: 0.3699 - val_auc: 0.8190
Epoch 10/150
93/93 [==============================] - 4s 41ms/step - loss: 5.5352 -
categorical_accuracy: 0.4325 - auc: 0.8417 - val_loss: 5.3792 -
val_categorical_accuracy: 0.4452 - val_auc: 0.8576
Epoch 11/150
93/93 [==============================] - 4s 41ms/step - loss: 5.3078 -
categorical_accuracy: 0.4389 - auc: 0.8597 - val_loss: 5.1681 -
val_categorical_accuracy: 0.4688 - val_auc: 0.8757
Epoch 12/150
93/93 [==============================] - 4s 41ms/step - loss: 5.0967 -
categorical_accuracy: 0.4680 - auc: 0.8747 - val_loss: 5.0065 -
val_categorical_accuracy: 0.5118 - val_auc: 0.8797
Epoch 13/150
93/93 [==============================] - 4s 41ms/step - loss: 4.8972 -
categorical_accuracy: 0.5013 - auc: 0.8865 - val_loss: 4.7970 -
val_categorical_accuracy: 0.5505 - val_auc: 0.8959
Epoch 14/150
93/93 [==============================] - 3s 29ms/step - loss: 4.7026 -
categorical_accuracy: 0.5272 - auc: 0.8981 - val_loss: 4.8751 -
val_categorical_accuracy: 0.4903 - val_auc: 0.8653
Epoch 15/150
93/93 [==============================] - 4s 41ms/step - loss: 4.5211 -
categorical_accuracy: 0.5524 - auc: 0.9081 - val_loss: 4.5490 -
val_categorical_accuracy: 0.5226 - val_auc: 0.8957
Epoch 16/150
93/93 [==============================] - 4s 41ms/step - loss: 4.3946 -
categorical_accuracy: 0.5498 - auc: 0.9094 - val_loss: 4.4190 -
val_categorical_accuracy: 0.5333 - val_auc: 0.8973
Epoch 17/150
93/93 [==============================] - 4s 40ms/step - loss: 4.2364 -
categorical_accuracy: 0.5605 - auc: 0.9144 - val_loss: 4.3062 -
val_categorical_accuracy: 0.5462 - val_auc: 0.8968
```

```
Epoch 18/150
93/93 [==============================] - 4s 41ms/step - loss: 4.0783 -
categorical_accuracy: 0.5670 - auc: 0.9215 - val_loss: 4.2471 -
val_categorical_accuracy: 0.5376 - val_auc: 0.8944
Epoch 19/150
93/93 [==============================] - 4s 41ms/step - loss: 3.9881 -
categorical_accuracy: 0.5729 - auc: 0.9186 - val_loss: 4.0957 -
val_categorical_accuracy: 0.5462 - val_auc: 0.8952
Epoch 20/150
93/93 [==============================] - 3s 29ms/step - loss: 3.9060 -
categorical_accuracy: 0.5740 - auc: 0.9142 - val_loss: 4.5532 -
val_categorical_accuracy: 0.3957 - val_auc: 0.7948
Epoch 21/150
93/93 [==============================] - 4s 41ms/step - loss: 3.6945 -
categorical_accuracy: 0.6095 - auc: 0.9294 - val_loss: 3.7862 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9099
Epoch 22/150
93/93 [==============================] - 4s 42ms/step - loss: 3.6114 -
categorical_accuracy: 0.6009 - auc: 0.9265 - val_loss: 3.7434 -
val_categorical_accuracy: 0.5699 - val_auc: 0.8998
Epoch 23/150
93/93 [==============================] - 4s 41ms/step - loss: 3.4872 -
categorical_accuracy: 0.6111 - auc: 0.9306 - val_loss: 3.6300 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9023
Epoch 24/150
93/93 [==============================] - 4s 41ms/step - loss: 3.3429 -
categorical_accuracy: 0.6089 - auc: 0.9369 - val_loss: 3.4766 -
val_categorical_accuracy: 0.5677 - val_auc: 0.9111
Epoch 25/150
93/93 [==============================] - 4s 43ms/step - loss: 3.2550 -
categorical_accuracy: 0.6218 - auc: 0.9349 - val_loss: 3.4446 -
val_categorical_accuracy: 0.5849 - val_auc: 0.9035
Epoch 26/150
93/93 [==============================] - 3s 30ms/step - loss: 3.1258 -
categorical_accuracy: 0.6412 - auc: 0.9406 - val_loss: 3.4508 -
val_categorical_accuracy: 0.5634 - val_auc: 0.8911
Epoch 27/150
93/93 [==============================] - 4s 42ms/step - loss: 3.0263 -
categorical_accuracy: 0.6471 - auc: 0.9412 - val_loss: 3.2324 -
val_categorical_accuracy: 0.5957 - val_auc: 0.9059
Epoch 28/150
93/93 [==============================] - 4s 48ms/step - loss: 2.9756 -
categorical_accuracy: 0.6396 - auc: 0.9364 - val_loss: 3.0750 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9165
Epoch 29/150
93/93 [==============================] - 4s 41ms/step - loss: 2.8283 -
categorical_accuracy: 0.6638 - auc: 0.9446 - val_loss: 2.9943 -
val_categorical_accuracy: 0.6086 - val_auc: 0.9189
```

```
Epoch 30/150
93/93 [==============================] - 4s 42ms/step - loss: 2.7451 -
categorical_accuracy: 0.6719 - auc: 0.9454 - val_loss: 2.9052 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9183
Epoch 31/150
93/93 [==============================] - 3s 29ms/step - loss: 2.6437 -
categorical_accuracy: 0.6600 - auc: 0.9478 - val_loss: 2.9429 -
val_categorical_accuracy: 0.5548 - val_auc: 0.9070
Epoch 32/150
93/93 [==============================] - 4s 42ms/step - loss: 2.5083 -
categorical_accuracy: 0.6756 - auc: 0.9553 - val_loss: 2.8347 -
val_categorical_accuracy: 0.5978 - val_auc: 0.9136
Epoch 33/150
93/93 [==============================] - 4s 42ms/step - loss: 2.4492 -
categorical_accuracy: 0.6848 - auc: 0.9529 - val_loss: 2.7185 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9151
Epoch 34/150
93/93 [==============================] - 3s 30ms/step - loss: 2.4409 -
categorical_accuracy: 0.6498 - auc: 0.9458 - val_loss: 2.9057 -
val_categorical_accuracy: 0.5355 - val_auc: 0.8774
Epoch 35/150
93/93 [==============================] - 4s 41ms/step - loss: 2.2896 -
categorical_accuracy: 0.6912 - auc: 0.9552 - val_loss: 2.6772 -
val_categorical_accuracy: 0.5957 - val_auc: 0.9086
Epoch 36/150
93/93 [==============================] - 3s 30ms/step - loss: 2.2168 -
categorical_accuracy: 0.6810 - auc: 0.9558 - val_loss: 2.8595 -
val_categorical_accuracy: 0.5204 - val_auc: 0.8787
Epoch 37/150
93/93 [==============================] - 4s 43ms/step - loss: 2.1791 -
categorical_accuracy: 0.6832 - auc: 0.9524 - val_loss: 2.4822 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9114
Epoch 38/150
93/93 [==============================] - 4s 41ms/step - loss: 2.1085 -
categorical_accuracy: 0.6724 - auc: 0.9537 - val_loss: 2.3329 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9217
Epoch 39/150
93/93 [==============================] - 4s 40ms/step - loss: 2.0427 -
categorical_accuracy: 0.6902 - auc: 0.9544 - val_loss: 2.3006 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9182
Epoch 40/150
93/93 [==============================] - 3s 30ms/step - loss: 1.9271 -
categorical_accuracy: 0.7047 - auc: 0.9602 - val_loss: 2.5675 -
val_categorical_accuracy: 0.5634 - val_auc: 0.8886
Epoch 41/150
93/93 [==============================] - 3s 29ms/step - loss: 1.9937 -
categorical_accuracy: 0.6654 - auc: 0.9464 - val_loss: 2.4117 -
val_categorical_accuracy: 0.5527 - val_auc: 0.8866
```

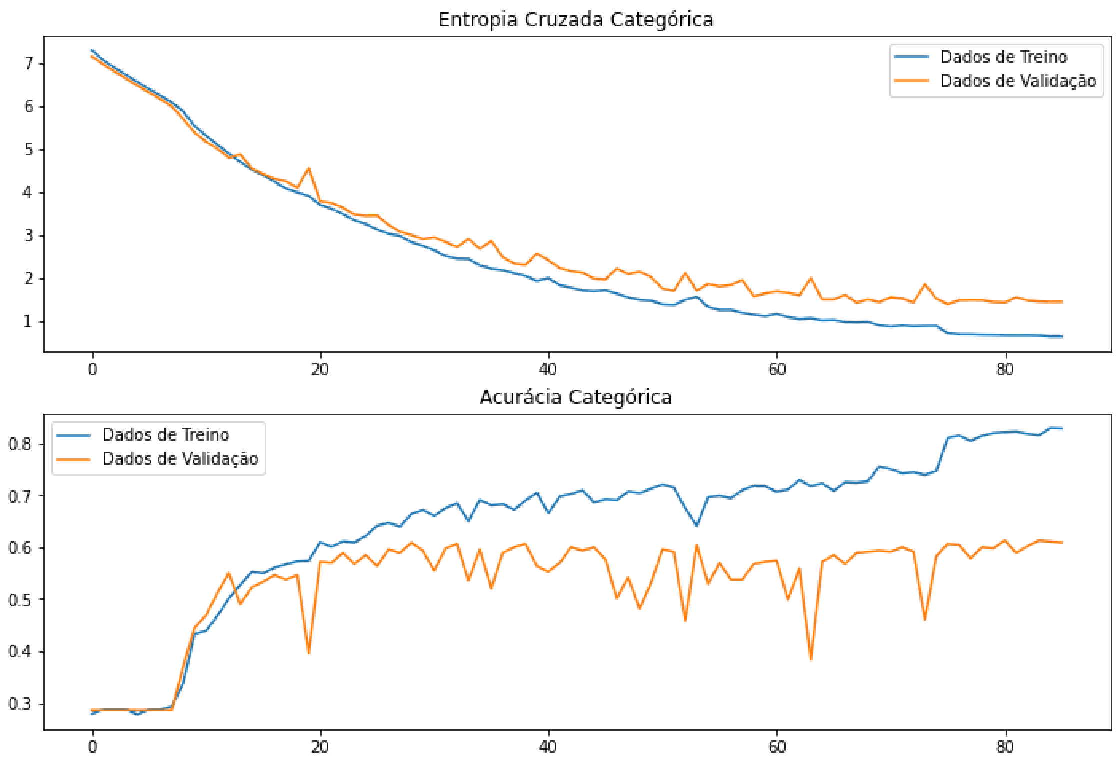
```
Epoch 42/150
93/93 [==============================] - 4s 44ms/step - loss: 1.8285 -
categorical_accuracy: 0.6977 - auc: 0.9593 - val_loss: 2.2297 -
val_categorical_accuracy: 0.5699 - val_auc: 0.9060
Epoch 43/150
93/93 [==============================] - 4s 41ms/step - loss: 1.7709 -
categorical_accuracy: 0.7025 - auc: 0.9593 - val_loss: 2.1573 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9140
Epoch 44/150
93/93 [==============================] - 4s 41ms/step - loss: 1.7049 -
categorical_accuracy: 0.7095 - auc: 0.9609 - val_loss: 2.1188 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9062
Epoch 45/150
93/93 [==============================] - 4s 42ms/step - loss: 1.6859 -
categorical_accuracy: 0.6864 - auc: 0.9582 - val_loss: 1.9811 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9203
Epoch 46/150
93/93 [==============================] - 4s 39ms/step - loss: 1.7096 -
categorical_accuracy: 0.6923 - auc: 0.9501 - val_loss: 1.9607 -
val_categorical_accuracy: 0.5763 - val_auc: 0.9172
Epoch 47/150
93/93 [==============================] - 3s 29ms/step - loss: 1.6324 -
categorical_accuracy: 0.6907 - auc: 0.9547 - val_loss: 2.2183 -
val_categorical_accuracy: 0.5011 - val_auc: 0.8721
Epoch 48/150
93/93 [==============================] - 3s 29ms/step - loss: 1.5425 -
categorical_accuracy: 0.7074 - auc: 0.9590 - val_loss: 2.0885 -
val_categorical_accuracy: 0.5419 - val_auc: 0.8845
Epoch 49/150
93/93 [==============================] - 3s 29ms/step - loss: 1.4870 -
categorical_accuracy: 0.7041 - auc: 0.9608 - val_loss: 2.1452 -
val_categorical_accuracy: 0.4817 - val_auc: 0.8698
Epoch 50/150
93/93 [==============================] - 3s 29ms/step - loss: 1.4699 -
categorical_accuracy: 0.7127 - auc: 0.9587 - val_loss: 2.0166 -
val_categorical_accuracy: 0.5312 - val_auc: 0.8823
Epoch 51/150
93/93 [==============================] - 4s 41ms/step - loss: 1.3844 -
categorical_accuracy: 0.7208 - auc: 0.9631 - val_loss: 1.7482 -
val_categorical_accuracy: 0.5957 - val_auc: 0.9198
Epoch 52/150
93/93 [==============================] - 4s 41ms/step - loss: 1.3696 -
categorical_accuracy: 0.7149 - auc: 0.9611 - val_loss: 1.6982 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9197
Epoch 53/150
93/93 [==============================] - 3s 30ms/step - loss: 1.4885 -
categorical_accuracy: 0.6751 - auc: 0.9447 - val_loss: 2.1110 -
val_categorical_accuracy: 0.4581 - val_auc: 0.8466
```

```
Epoch 54/150
93/93 [==============================] - 3s 29ms/step - loss: 1.5591 -
categorical_accuracy: 0.6407 - auc: 0.9338 - val_loss: 1.7012 -
val_categorical_accuracy: 0.6043 - val_auc: 0.9109
Epoch 55/150
93/93 [==============================] - 3s 29ms/step - loss: 1.3199 -
categorical_accuracy: 0.6966 - auc: 0.9573 - val_loss: 1.8573 -
val_categorical_accuracy: 0.5290 - val_auc: 0.8943
Epoch 56/150
93/93 [==============================] - 3s 30ms/step - loss: 1.2533 -
categorical_accuracy: 0.6993 - auc: 0.9605 - val_loss: 1.8006 -
val_categorical_accuracy: 0.5699 - val_auc: 0.9003
Epoch 57/150
93/93 [==============================] - 3s 29ms/step - loss: 1.2544 -
categorical_accuracy: 0.6945 - auc: 0.9584 - val_loss: 1.8327 -
val_categorical_accuracy: 0.5376 - val_auc: 0.8848
Epoch 58/150
93/93 [==============================] - 3s 29ms/step - loss: 1.1883 -
categorical_accuracy: 0.7101 - auc: 0.9620 - val_loss: 1.9495 -
val_categorical_accuracy: 0.5376 - val_auc: 0.8881
Epoch 59/150
93/93 [==============================] - 4s 42ms/step - loss: 1.1431 -
categorical_accuracy: 0.7181 - auc: 0.9642 - val_loss: 1.5662 -
val_categorical_accuracy: 0.5677 - val_auc: 0.9171
Epoch 60/150
93/93 [==============================] - 3s 29ms/step - loss: 1.1077 -
categorical_accuracy: 0.7176 - auc: 0.9654 - val_loss: 1.6377 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9013
Epoch 61/150
93/93 [==============================] - 3s 29ms/step - loss: 1.1606 -
categorical_accuracy: 0.7068 - auc: 0.9571 - val_loss: 1.6862 -
val_categorical_accuracy: 0.5742 - val_auc: 0.8995
Epoch 62/150
93/93 [==============================] - 3s 29ms/step - loss: 1.0889 -
categorical_accuracy: 0.7111 - auc: 0.9633 - val_loss: 1.6513 -
val_categorical_accuracy: 0.4989 - val_auc: 0.8979
Epoch 63/150
93/93 [==============================] - 3s 30ms/step - loss: 1.0411 -
categorical_accuracy: 0.7294 - auc: 0.9662 - val_loss: 1.5929 -
val_categorical_accuracy: 0.5591 - val_auc: 0.9010
Epoch 64/150
93/93 [==============================] - 3s 30ms/step - loss: 1.0618 -
categorical_accuracy: 0.7176 - auc: 0.9626 - val_loss: 1.9955 -
val_categorical_accuracy: 0.3828 - val_auc: 0.8523
Epoch 65/150
93/93 [==============================] - 4s 41ms/step - loss: 1.0115 -
categorical_accuracy: 0.7230 - auc: 0.9658 - val_loss: 1.4960 -
val_categorical_accuracy: 0.5720 - val_auc: 0.9107
```

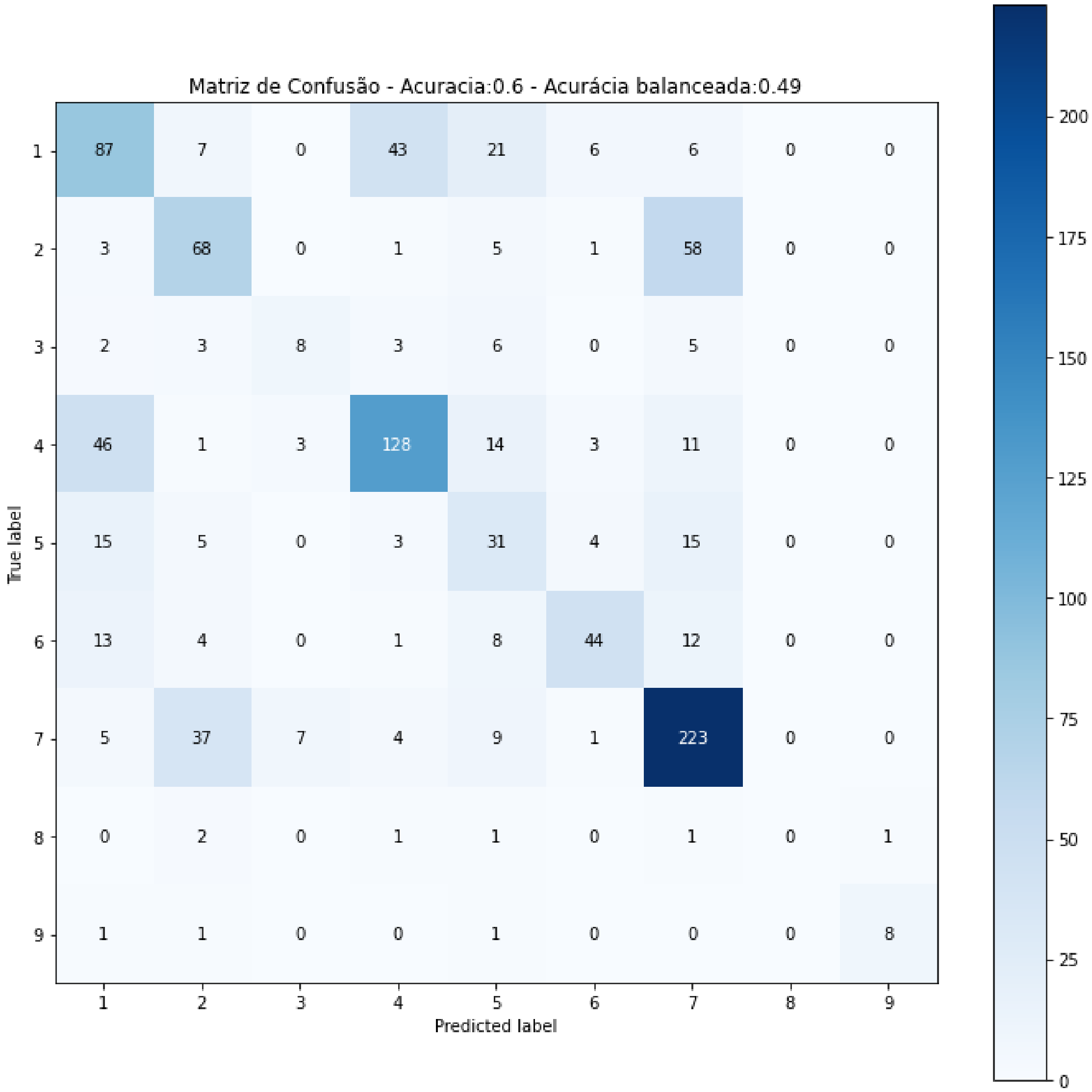
```
Epoch 66/150
93/93 [==============================] - 4s 42ms/step - loss: 1.0223 -
categorical_accuracy: 0.7084 - auc: 0.9636 - val_loss: 1.4944 -
val_categorical_accuracy: 0.5849 - val_auc: 0.9143
Epoch 67/150
93/93 [==============================] - 3s 29ms/step - loss: 0.9746 -
categorical_accuracy: 0.7251 - auc: 0.9664 - val_loss: 1.6026 -
val_categorical_accuracy: 0.5677 - val_auc: 0.9040
Epoch 68/150
93/93 [==============================] - 4s 40ms/step - loss: 0.9673 -
categorical_accuracy: 0.7240 - auc: 0.9665 - val_loss: 1.4206 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9181
Epoch 69/150
93/93 [==============================] - 3s 30ms/step - loss: 0.9768 -
categorical_accuracy: 0.7267 - auc: 0.9651 - val_loss: 1.4985 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9127
Epoch 70/150
93/93 [==============================] - 3s 30ms/step - loss: 0.8942 -
categorical_accuracy: 0.7552 - auc: 0.9707 - val_loss: 1.4345 -
val_categorical_accuracy: 0.5935 - val_auc: 0.9150
Epoch 71/150
93/93 [==============================] - 3s 29ms/step - loss: 0.8658 -
categorical_accuracy: 0.7504 - auc: 0.9733 - val_loss: 1.5455 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9083
Epoch 72/150
93/93 [==============================] - 3s 29ms/step - loss: 0.8887 -
categorical_accuracy: 0.7423 - auc: 0.9702 - val_loss: 1.5170 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9138
Epoch 73/150
93/93 [==============================] - 3s 29ms/step - loss: 0.8713 -
categorical_accuracy: 0.7445 - auc: 0.9723 - val_loss: 1.4264 -
val_categorical_accuracy: 0.5914 - val_auc: 0.9118
Epoch 74/150
93/93 [==============================] - 3s 29ms/step - loss: 0.8787 -
categorical_accuracy: 0.7391 - auc: 0.9715 - val_loss: 1.8513 -
val_categorical_accuracy: 0.4602 - val_auc: 0.8717
Epoch 75/150
92/93 [============================>.] - ETA: 0s - loss: 0.8803 -
categorical_accuracy: 0.7467 - auc: 0.9713
Epoch 00075: ReduceLROnPlateau reducing learning rate to 0.005000000074505806.
93/93 [==============================] - 3s 29ms/step - loss: 0.8815 -
categorical_accuracy: 0.7466 - auc: 0.9712 - val_loss: 1.5150 -
val_categorical_accuracy: 0.5828 - val_auc: 0.9074
Epoch 76/150
93/93 [==============================] - 4s 43ms/step - loss: 0.7096 -
categorical_accuracy: 0.8107 - auc: 0.9839 - val_loss: 1.3872 -
val_categorical_accuracy: 0.6065 - val_auc: 0.9189
Epoch 77/150
```

```
93/93 [==============================] - 3s 29ms/step - loss: 0.6849 -
categorical_accuracy: 0.8150 - auc: 0.9854 - val_loss: 1.4791 -
val_categorical_accuracy: 0.6043 - val_auc: 0.9164
Epoch 78/150
93/93 [==============================] - 3s 30ms/step - loss: 0.6837 -
categorical_accuracy: 0.8047 - auc: 0.9849 - val_loss: 1.4842 -
val_categorical_accuracy: 0.5785 - val_auc: 0.9122
Epoch 79/150
93/93 [==============================] - 3s 29ms/step - loss: 0.6735 -
categorical_accuracy: 0.8144 - auc: 0.9857 - val_loss: 1.4817 -
val_categorical_accuracy: 0.6000 - val_auc: 0.9159
Epoch 80/150
93/93 [==============================] - 3s 29ms/step - loss: 0.6703 -
categorical_accuracy: 0.8198 - auc: 0.9861 - val_loss: 1.4389 -
val_categorical_accuracy: 0.5978 - val_auc: 0.9162
Epoch 81/150
93/93 [==============================] - 3s 30ms/step - loss: 0.6641 -
categorical_accuracy: 0.8214 - auc: 0.9863 - val_loss: 1.4264 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9181
Epoch 82/150
93/93 [==============================] - 3s 29ms/step - loss: 0.6625 -
categorical_accuracy: 0.8225 - auc: 0.9863 - val_loss: 1.5444 -
val_categorical_accuracy: 0.5892 - val_auc: 0.9114
Epoch 83/150
93/93 [==============================] - ETA: 0s - loss: 0.6640 -
categorical_accuracy: 0.8182 - auc: 0.9862
Epoch 00083: ReduceLROnPlateau reducing learning rate to 0.0004999999888241291.
93/93 [==============================] - 3s 29ms/step - loss: 0.6640 -
categorical_accuracy: 0.8182 - auc: 0.9862 - val_loss: 1.4737 -
val_categorical_accuracy: 0.6022 - val_auc: 0.9162
Epoch 84/150
93/93 [==============================] - 3s 29ms/step - loss: 0.6578 -
categorical_accuracy: 0.8155 - auc: 0.9861 - val_loss: 1.4513 -
val_categorical_accuracy: 0.6129 - val_auc: 0.9175
Epoch 85/150
93/93 [==============================] - 3s 29ms/step - loss: 0.6373 -
categorical_accuracy: 0.8295 - auc: 0.9875 - val_loss: 1.4422 -
val_categorical_accuracy: 0.6108 - val_auc: 0.9186
Epoch 86/150
93/93 [==============================] - 3s 29ms/step - loss: 0.6364 -
categorical_accuracy: 0.8284 - auc: 0.9876 - val_loss: 1.4413 -
val_categorical_accuracy: 0.6086 - val_auc: 0.9180
Epoch 00086: early stopping
```

[61]:
```
modelo3_3_history = json.load(open("./modelos/history_modelo3_3.json",'r'))
plot_treinamento(modelo3_3_history)
```



[62]:
```
plot_matriz_confusao(modelo3_3,X_test_novo,Y_test_novo)
```



2.11 Avaliando as versões do modelo 3 juntas

[74]:
```
reset_keras()
```

129

[75]:
```
modelo3 = load_model("./modelos/modelo3.hdf5")
modelo3_1 = load_model("./modelos/modelo3_1.hdf5")
modelo3_2 = load_model("./modelos/modelo3_2.hdf5")
modelo3_3 = load_model("./modelos/modelo3_3.hdf5")
```

[79]:
```
scores_modelos3 =    {"Acuracia_balanceada":[balanced_accuracy_score(y_pred=tf.
↪argmax(modelagem.predict(X_test_novo), axis = 1),
                                                                    ␣
↪y_true=Y_test_novo.argmax(axis=1)) for modelagem in
                                             ␣
↪[modelo3,modelo3_1,modelo3_2,modelo3_3]],
                       "Acuracia": [accuracy_score(y_pred=tf.argmax(modelagem.
↪predict(X_test_novo), axis = 1),
                                                  y_true=Y_test_novo.
↪argmax(axis=1)) for modelagem in
                                             ␣
↪[modelo3,modelo3_1,modelo3_2,modelo3_3]],
                       "F1": [f1_score(y_pred=tf.argmax(modelagem.
↪predict(X_test_novo), axis = 1),
                                      y_true=Y_test_novo.argmax(axis=1),
                                     average = "macro") for modelagem in
                                             ␣
↪[modelo3,modelo3_1,modelo3_2,modelo3_3]],
                      "Modelo": ["modelo 3", "modelo 3.1","modelo 3.2", "modelo␣
↪3.3"]
                      }
```

[80]:
```
pd.DataFrame(scores_modelos3)
```

[80]:

	Acuracia_balanceada	Acuracia	F1	Modelo
0	0.404090	0.599799	0.438169	modelo 3
1	0.472758	0.582748	0.493632	modelo 3.1
2	0.500217	0.595787	0.523588	modelo 3.2
3	0.479073	0.580742	0.493529	modelo 3.3

[ ]: