tropical_cyclone_estimation

August 1, 2023

1 Estimação da Intensidade de Ciclones Tropicais

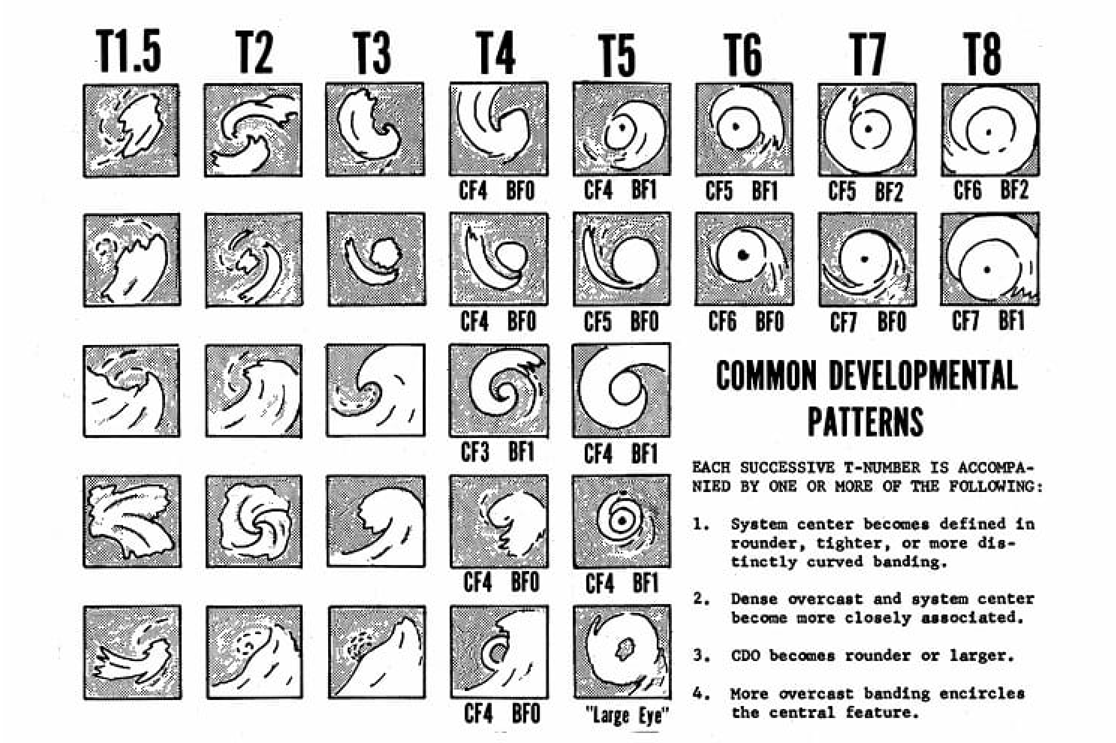
1.1 Introdução

Ciclones tropicais são sistemas de tempestades intensos que se originam nos oceanos em regiões próximas ao equador. Eles são caracterizados por ventos fortes, chuvas intensas e uma área de baixa pressão no centro, muitas vezes referida como o "olho" do ciclone. Os ciclones tropicais podem causar devastação em áreas costeiras, levando a inundações, danos a construções e perda de vidas. A monitorização e previsão desses fenômenos são vitais para a preparação e resposta adequadas a essas tempestades poderosas.

Dessa forma, uma vez que os furacões (ou ciclones tropicais) possuem ameaças substanciais e causam danos significativos a vidas e propriedades, estudar as etapas de um furacão é essencial para determinar seu impacto.

Porém, a análise de ciclones tropicais por meio somente de imagens de satélite não é suficiente, visto que a categoria desses é fortemente baseada na velocidade máxima dos ventos.

Sendo assim, para o problema de classificação de ciclones é utilizado a técnica Dvorak, que consiste em um algoritmo manual executado por um especialista na área que visa estimar a intensidade do ciclone com base em imagens de satélite.



Infelizmente, por se tratar de um algoritmo manual, este é muito suscetível a erros. Dessa forma, se faz necessário a construção de um modelo que elimine erros humanos na classificação de ciclones tropicais.

As Redes Neurais Convolucionais (CNNs) são excelentes para a classificação de imagens. Elas utilizam camadas especiais que podem identificar padrões nas imagens, como bordas e formas, tornando-as altamente eficazes em distinguir diferentes categorias visuais. Essa habilidade faz das CNNs uma ferramenta popular e poderosa na análise de imagem.

Portanto, a abordagem de Redes Neurais Convolucionais - pela sua natureza - é ideal para a construção de um modelo de aprendizagem de máquina que seja eficaz no problema de classificações de ciclones tropicais.

1.2 Objetivos

Dessa forma, o presente trabalho tem como a classificação Multi-Classe de ciclones tropicais, utilizando imagens de satélite junto com dados de intensidade:

- NC (No Category)
- TD (Tropical Depression)
- TS (Topical Storm)
- H1 (Category One)
- H2 (Category Two)
- H3 (Category Three)
- H4 (Category Four)
- H5 (Category Five)

1.3 Materiais e métodos

O presente trabalho foi iniciado no Bootcamp *HPC para IA*, realizado no *Laboratório Nacional de Computação Científica (LNCC)* em parceria com a *NVIDIA*. Nesse bootcamp, foi feito uma releitura do modelo descrito no Artigo de Pesquisa intitulado "Estimação da Intensidade do Ciclone Tropical Usando uma Rede Neural Convolucional Profunda" * por Ritesh Pradhan, Ramazan S. Aygun, Membro Sênior, IEEE, Manil Maskey, Membro, IEEE, Rahul Ramachandran, Membro Sênior, IEEE, e Daniel J. Cecil *

O modelo original, utiliza o *Caffe Framework* e possui a seguinte arquitetura:

Assim, um modelo do tipo CNN utilizando o framework Keras foi desenvolvido baseando-se na arquitetura mostrada no artigo. O modelo utiliza imagens de ciclones tropicais anotadas com suas respectivas intensidades como entrada e tem como objetivo prever a intensidade de futuras imagens de satélite de ciclones tropicais.

1.4 Configuração Do Ambiente

Bibliotecas utilizadas:

[1]:
```
!pip install gdown
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages
(4.6.6)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from gdown) (3.12.2)
Requirement already satisfied: requests[socks] in
/usr/local/lib/python3.10/dist-packages (from gdown) (2.27.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from gdown) (4.65.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from gdown) (4.11.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-
packages (from beautifulsoup4->gdown) (2.4.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown)
(2023.7.22)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
```

[2]:
```
from __future__ import absolute_import, division, print_function,␣
 ↪unicode_literals
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import gdown
import cv2
import os
from datetime import datetime
import pandas as pd
from scipy import interpolate
import matplotlib.pyplot as plt
import random
from collections import Counter
import seaborn as sn
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten ,Dropout, MaxPooling2D
from keras import backend as K
import functools
from functools import reduce
```

Checagem de GPUs disponíveis

[3]:
```
print(tf.__version__)
tf.test.gpu_device_name()
```

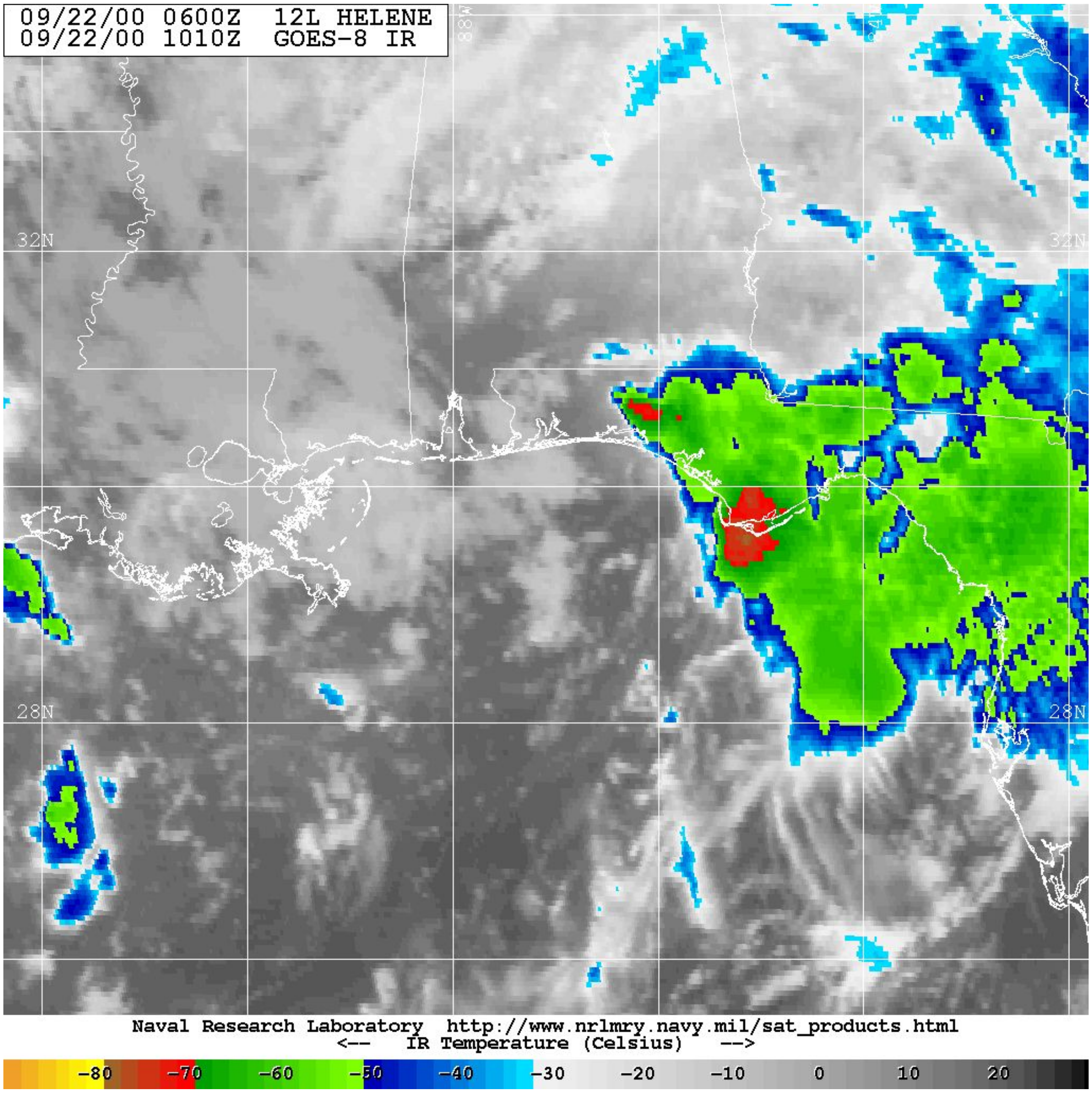
```
2.12.0
```

[3]:
```
'/device:GPU:0'
```

1.5 Dataset

O dataset é constituído de dois componentes:

- Imagens de satélite de ciclones tropicais, fornecidas pela marinha americana.



- Dados CSV provenientes da base hurdat2, contendo diversos dados sobre ciclones tropicais (dentre eles a classe do método Dvorak).

```
,index,id,name,date,record_identifier,status_of_system,latitude,longitude,maximum_sustained_wi
0,0,AL011851,UNNAMED,1851-06-25 00:00:00,,HU,28.0N,94.8W,80,,,,,,,,,,,,,
1,1,AL011851,UNNAMED,1851-06-25 06:00:00,,HU,28.0N,95.4W,80,,,,,,,,,,,,,

...
```

[4]:
```
url_images = 'https://drive.google.com/file/d/1-GaxHdsFv_gNmFqSxz1wRhc3a9uwMgsZ/
 ↪view?usp=sharing'
output_images = 'dataset.zip'

url_text = 'https://drive.google.com/file/d/1qs5c-uDTcc-RBTKqHuaQBdMpeEMjpCRn/
 ↪view?usp=sharing'
output_text = 'atlantic_storms.csv'
```

```
gdown.cached_download(url_images,
                      output_images,
                      quiet=False,
                      proxy=None,
                      fuzzy=True,
                      postprocess=gdown.extractall)

gdown.cached_download(url_text,
                      output_text,
                      quiet=False,
                      proxy=None,
                      fuzzy=True)
```

```
Cached Downloading: dataset.zip
Downloading…
From: https://drive.google.com/uc?id=1-GaxHdsFv_gNmFqSxz1wRhc3a9uwMgsZ
To: /root/.cache/gdown/tmp6nnx51ap/dl
100%|      | 1.11G/1.11G [00:15<00:00, 73.3MB/s]
Cached Downloading: atlantic_storms.csv
Downloading…
From: https://drive.google.com/uc?id=1qs5c-uDTcc-RBTKqHuaQBdMpeEMjpCRn
To: /root/.cache/gdown/tmpk6knkjn_/dl
100%|      | 4.53M/4.53M [00:00<00:00, 190MB/s]
```

[4]: 'atlantic_storms.csv'

[5]:
```
dir = 'Dataset/'
a = os.listdir(dir)
a = filter(lambda x: x != 'Aug', a)

total = [
    file
    for i in a
    for j in os.listdir(dir + i)
    for k in os.listdir(dir + i + '/' + j)
    for l in os.listdir(dir + i + '/' + j + '/' + k)
    for file in os.listdir(dir + i + '/' + j + '/' + k + '/' + l + '/ir/geo/
 ↪1km')
]

for i in a:
    for j in os.listdir(dir + i):
        for k in os.listdir(dir + i + '/' + j):
            for l in os.listdir(dir + i + '/' + j + '/' + k):
```

```
                e = os.listdir(dir + i + '/' + j + '/' + k + '/' + l + '/ir/geo/
 ↪1km')
                print(j + '-> ' + l + ' --> ' + str(len(e)))

print('Total number of images present in the Dataset :', len(total))
```

```
Total number of images present in the Dataset : 32611
```

[6]:
```
print('Total number of rows present in the Text Dataset:', len(pd.
 ↪read_csv('atlantic_storms.csv')))
```

```
Total number of rows present in the Text Dataset: 51310
```

Dessa forma, o dataset de imagens será anotado com as intensidades dos ciclones obtidas do dataset de texto, a fim de alimentar o modelo de machine learning.

1.6 Pré Processamento e conjuntos de dados

1.6.1 Pré Processamento

A base de dados de imagens contem imagens de ciclones amostrada a cada 2 horas. Já a base de dados de texto, possui dados desses ciclones amostrados a cada 6 horas. Dessa forma, a função *load_dataset* carrega as imagens, combina com os dados da base de texto e interpola os dados que faltam. Além disso, a função recebe outra função para realizar o *data augmentation*, etapa que será feita posteriormente.

[7]:
```
def dummy():
  pass

def load_dataset(augment_fn=dummy):
    filenames = []
    labels = []
    i = 0
    df = pd.read_csv('atlantic_storms.csv')
    dir = 'Dataset/tcdat/'
    a = os.listdir(dir)
    file_path = "Dataset/Aug/"
    directory = os.path.dirname(file_path)

    try:
        os.stat(directory)
    except:
        os.mkdir(directory)
    aug = 0
    for j in a:
        c = os.listdir(dir + '/' + j)
        for k in c:
            d = os.listdir(dir + '/' + j + '/' + k)
            for l in d:
```

```
                print('.', end='')
                start_year = '20' + j[2:] + '-01-01'
                end_year = '20' + j[2:] + '-12-31'
                cyc_name = l[4:]
                mask = (df['date'] > start_year) & (df['date'] <= end_year) & (
                    df['name'] == cyc_name)
                cyc_pd = df.loc[mask]
                first = (datetime.strptime(cyc_pd['date'].iloc[0],
                                           "%Y-%m-%d %H:%M:%S"))
                last = (datetime.strptime(cyc_pd['date'].iloc[-1],
                                          "%Y-%m-%d %H:%M:%S"))
                text_time = []
                text_vel = []
                for q in range(len(cyc_pd['date'])):
                    text_vel.append(
                        cyc_pd['maximum_sustained_wind_knots'].iloc[q])
                    text_time.append((datetime.strptime(
                        cyc_pd['date'].iloc[q], "%Y-%m-%d %H:%M:%S") -
                                      first).total_seconds())
                func = interpolate.splrep(text_time, text_vel)
                e = os.listdir(dir + '/' + j + '/' + k + '/' + l +
                               '/ir/geo/1km')
                e.sort()
                for m in e:
                    try:
                        time = (datetime.strptime(m[:13], "%Y%m%d.%H%M"))
                        name = dir + j + '/' + k + '/' + l + '/ir/geo/1km/' + m
                        if (time > first and time < last):
                            val = int(
                                interpolate.splev(
                                    (time - first).total_seconds(), func))
                            filenames.append(name)
                            if val <= 20:
                                labels.append(0)
                            elif val > 20 and val <= 33:
                                labels.append(1)
                            elif val > 33 and val <= 63:
                                labels.append(2)
                            elif val > 63 and val <= 82:
                                labels.append(3)
                            elif val > 82 and val <= 95:
                                labels.append(4)
                            elif val > 95 and val <= 112:
                                labels.append(5)
                            elif val > 112 and val <= 136:
                                labels.append(6)
                            elif val > 136:
```

```
                                labels.append(7)
                            i = augment_fn(name, labels[-1], filenames, labels,
                                           i)
                    except:
                        pass
    print('')
    print(len(filenames))
    # Shuffle The Data
    # Zip Images with Appropriate Labels before Shuffling
    c = list(zip(filenames, labels))
    random.shuffle(c)
    #Unzip the Data Post Shuffling
    filenames, labels = zip(*c)
    filenames = list(filenames)
    labels = list(labels)
    return filenames, labels

def parse_function(filename, label):
    image_string = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image_string, channels=3)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, [232, 232])
    return image, label


def make_dataset(train_in, test_in, val_in):
    train = tf.data.Dataset.from_tensor_slices((train_in[0], train_in[1]))
    train = train.shuffle(len(train_in[0]))
    train = train.map(parse_function, num_parallel_calls=8)
    train = train.batch(train_in[2])
    train = train.prefetch(1)
    test = tf.data.Dataset.from_tensor_slices((test_in[0], test_in[1]))
    test = test.shuffle(len(test_in[0]))
    test = test.map(parse_function, num_parallel_calls=8)
    test = test.batch(test_in[2])
    test = test.prefetch(1)
    val = tf.data.Dataset.from_tensor_slices((val_in[0], val_in[1]))
    val = val.map(parse_function, num_parallel_calls=8)
    val = val.batch(val_in[2])
    val = val.prefetch(1)
    return train, test, val
```

[8]:
```
# Load dataset
filenames,labels = load_dataset()
```

…
27582

1.7 Conjuntos de dados

O dataset é dividido em:

- Treinamento: 70%
- Teste: 20%
- Validação: 10%

[9]:
```
def make_test_set(filenames, labels, val=0.1):
    classes = 8
    j = 0
    val_filenames = []
    val_labels = []
    new = [int(val * len(filenames) / classes)] * classes
    print(new)
    try:
        for i in range(len(filenames)):
            if (new[labels[i]] > 0):
                val_filenames.append(filenames[i])
                val_labels.append(labels[i])
                new[labels[i]] = new[labels[i]] - 1
                del filenames[i]
                del labels[i]
    except:
        pass

    c = list(zip(val_filenames, val_labels))
    random.shuffle(c)
    val_filenames, val_labels = zip(*c)
    val_filenames = list(val_filenames)
    val_labels = list(val_labels)
    print(Counter(labels))
    return val_filenames, val_labels
```

Divisão de dados de treino em cada classe:

[10]:
```
val_filenames , val_labels = make_test_set(filenames,labels,val=0.1)
```

```
[344, 344, 344, 344, 344, 344, 344, 344]
Counter({2: 7936, 3: 5339, 1: 3803, 4: 2934, 5: 2336, 6: 2178, 7: 204, 0: 100})
```

[11]:
```
test = 0.2
x_train, x_test, y_train, y_test = train_test_split(filenames, labels,␣
 ↪test_size=test, random_state=1)
```

O método de *Encoding* utilizado é o *One Hot Encoding*:

```
2 --- > [ 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0]

4 --- > [ 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0]
```

[12]:
```
y_train = tf.one_hot(y_train,depth=8)
y_test = tf.one_hot(y_test,depth=8)
val_labels = tf.one_hot(val_labels,depth=8)
```

[13]:
```
train,test,val =␣
↪make_dataset((x_train,y_train,128),(x_test,y_test,32),(val_filenames,val_labels,32))
```

1.8 Modelo v1: Arquitetura original

Os parâmetros das camadas, como as funções de ativação e o número de neurônios foram retirados do artigo original.

- `Conv2D(64, kernel_size=10, strides=3, activation='relu', input_shape=(232,232,3))`: Camada convolucional com 64 filtros, kernel de tamanho 10, stride de 3, ativação ReLU, e forma de entrada 232x232x3.
- `MaxPooling2D(pool_size=(3, 3), strides=2)`: Camada de pooling com tamanho de pool 3x3 e stride de 2.
- `Conv2D(256, kernel_size=5, strides=1, activation='relu')`: Camada convolucional com 256 filtros, kernel de tamanho 5, stride de 1, e ativação ReLU.
- `MaxPooling2D(pool_size=(3, 3), strides=2)`: Camada de pooling com tamanho de pool 3x3 e stride de 2.
- `Conv2D(288, kernel_size=3, strides=1, padding='same', activation='relu')`: Camada convolucional com 288 filtros, kernel de tamanho 3, stride de 1, padding do mesmo tamanho, e ativação ReLU.
- `MaxPooling2D(pool_size=(2, 2), strides=1)`: Camada de pooling com tamanho de pool 2x2 e stride de 1.
- `Conv2D(272, kernel_size=3, strides=1, padding='same', activation='relu')`: Camada convolucional com 272 filtros, kernel de tamanho 3, stride de 1, padding do mesmo tamanho, e ativação ReLU.
- `Conv2D(256, kernel_size=3, strides=1, activation='relu')`: Camada convolucional com 256 filtros, kernel de tamanho 3, stride de 1, e ativação ReLU.
- `MaxPooling2D(pool_size=(3, 3), strides=2)`: Camada de pooling com tamanho de pool 3x3 e stride de 2.
- `Dropout(0.5)`: Camada de dropout com taxa de 0.5 para regularização.
- `Flatten()`: Camada para achatamento dos dados.
- `Dense(3584, activation='relu')`: Camada densa com 3584 neurônios e ativação ReLU.
- `Dense(2048, activation='relu')`: Camada densa com 2048 neurônios e ativação ReLU.
- `Dense(8, activation='softmax')`: Camada de saída densa com 8 neurônios e ativação softmax para classificação.

[14]:
```
os.environ["CUDA_VISIBLE_DEVICES"]="0"
tf.random.set_seed(1337)

#Reset Graphs and Create Sequential model
K.clear_session()
model = Sequential()

#Convolution Layers
```

```python
model.add(Conv2D(64, kernel_size=10,strides=3, activation='relu',␣
  ↪input_shape=(232,232,3)))
model.add(MaxPooling2D(pool_size=(3, 3),strides=2))
model.add(Conv2D(256, kernel_size=5,strides=1,activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3),strides=2))
model.add(Conv2D(288, kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=1))
model.add(Conv2D(272, kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(Conv2D(256, kernel_size=3,strides=1,activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3),strides=2))
model.add(Dropout(0.5))
model.add(Flatten())

#Linear Layers
model.add(Dense(3584,activation='relu'))
model.add(Dense(2048,activation='relu'))
model.add(Dense(8, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 75, 75, 64)        19264

 max_pooling2d (MaxPooling2D  (None, 37, 37, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 33, 33, 256)       409856

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 16, 16, 288)       663840

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 288)      0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 272)       705296

 conv2d_4 (Conv2D)           (None, 13, 13, 256)       626944

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 256)        0
 2D)

 dropout (Dropout)           (None, 6, 6, 256)         0
```

```
 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 3584)              33033728

 dense_1 (Dense)             (None, 2048)              7342080

 dense_2 (Dense)             (None, 8)                 16392

=================================================================
Total params: 42,817,400
Trainable params: 42,817,400
Non-trainable params: 0
_________________________________________________________________
```

Os pesos iniciais do modelo foram computados e salvos. Isso vai deixar futuras execuções mais rápidas:

[15]:
```python
url_weights = 'https://drive.google.com/file/d/
 ↪1Csj_YFAXHtnGpOutxZ9A6CBrGAsZkpXu/view?usp=sharing'
output_weights = 'model.h5'

gdown.cached_download(url_weights,
                      output_weights,
                      quiet=False,
                      proxy=None,
                      fuzzy=True)
```

```
Cached Downloading: model.h5
Downloading…
From: https://drive.google.com/uc?id=1Csj_YFAXHtnGpOutxZ9A6CBrGAsZkpXu
To: /root/.cache/gdown/tmpdcq61y5a/dl
100%|      | 343M/343M [00:04<00:00, 84.5MB/s]
```

[15]: 'model.h5'

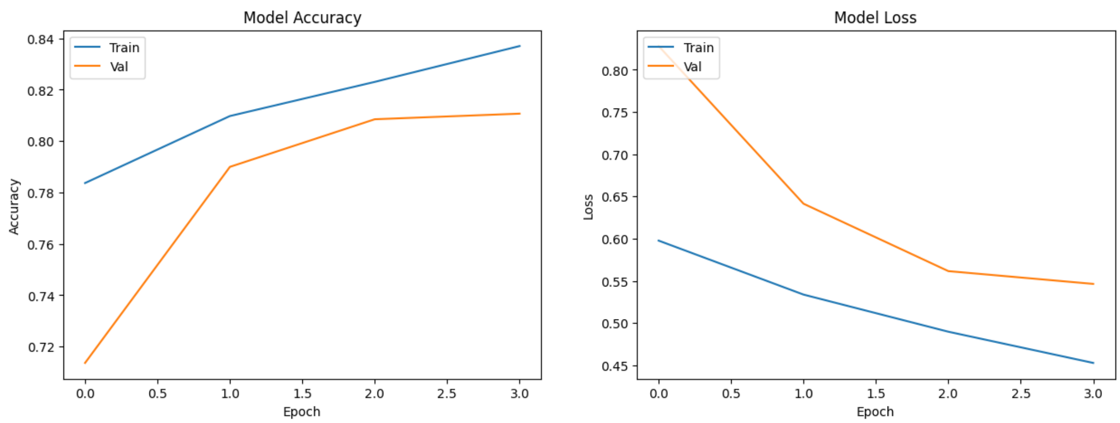
[16]:
```python
top2_acc = functools.partial(keras.metrics.top_k_categorical_accuracy, k=2)
top2_acc.__name__ = 'top2_acc'

epochs = 4
model.load_weights("model.h5")

# Optimizer
sgd = keras.optimizers.legacy.SGD(learning_rate=0.001, decay=1e-6, momentum=0.9)

#Compile Model with Loss Function , Optimizer and Metrics
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=sgd,
              metrics=['accuracy',top2_acc])
```

```
# Train the Model
trained_model = model.fit(train,
          epochs=epochs,
          verbose=1,
          validation_data=val)

# Test Model Aganist Validation Set
score = model.evaluate(test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Epoch 1/4
156/156 [==============================] - 44s 190ms/step - loss: 0.5977 -
accuracy: 0.7837 - top2_acc: 0.9222 - val_loss: 0.8281 - val_accuracy: 0.7137 -
val_top2_acc: 0.8964
Epoch 2/4
156/156 [==============================] - 31s 196ms/step - loss: 0.5338 -
accuracy: 0.8098 - top2_acc: 0.9350 - val_loss: 0.6414 - val_accuracy: 0.7900 -
val_top2_acc: 0.9150
Epoch 3/4
156/156 [==============================] - 31s 200ms/step - loss: 0.4898 -
accuracy: 0.8230 - top2_acc: 0.9451 - val_loss: 0.5616 - val_accuracy: 0.8085 -
val_top2_acc: 0.9382
Epoch 4/4
156/156 [==============================] - 29s 182ms/step - loss: 0.4529 -
accuracy: 0.8370 - top2_acc: 0.9509 - val_loss: 0.5465 - val_accuracy: 0.8107 -
val_top2_acc: 0.9353
Test loss: 0.3319806158542633
Test accuracy: 0.8848167657852173
```

[17]:
```
f = plt.figure(figsize=(15,5))
ax = f.add_subplot(121)
ax.plot(trained_model.history['accuracy'])
ax.plot(trained_model.history['val_accuracy'])
ax.set_title('Model Accuracy')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.legend(['Train', 'Val'])

ax2 = f.add_subplot(122)
ax2.plot(trained_model.history['loss'])
ax2.plot(trained_model.history['val_loss'])
ax2.set_title('Model Loss')
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend(['Train', 'Val'],loc= 'upper left')
```

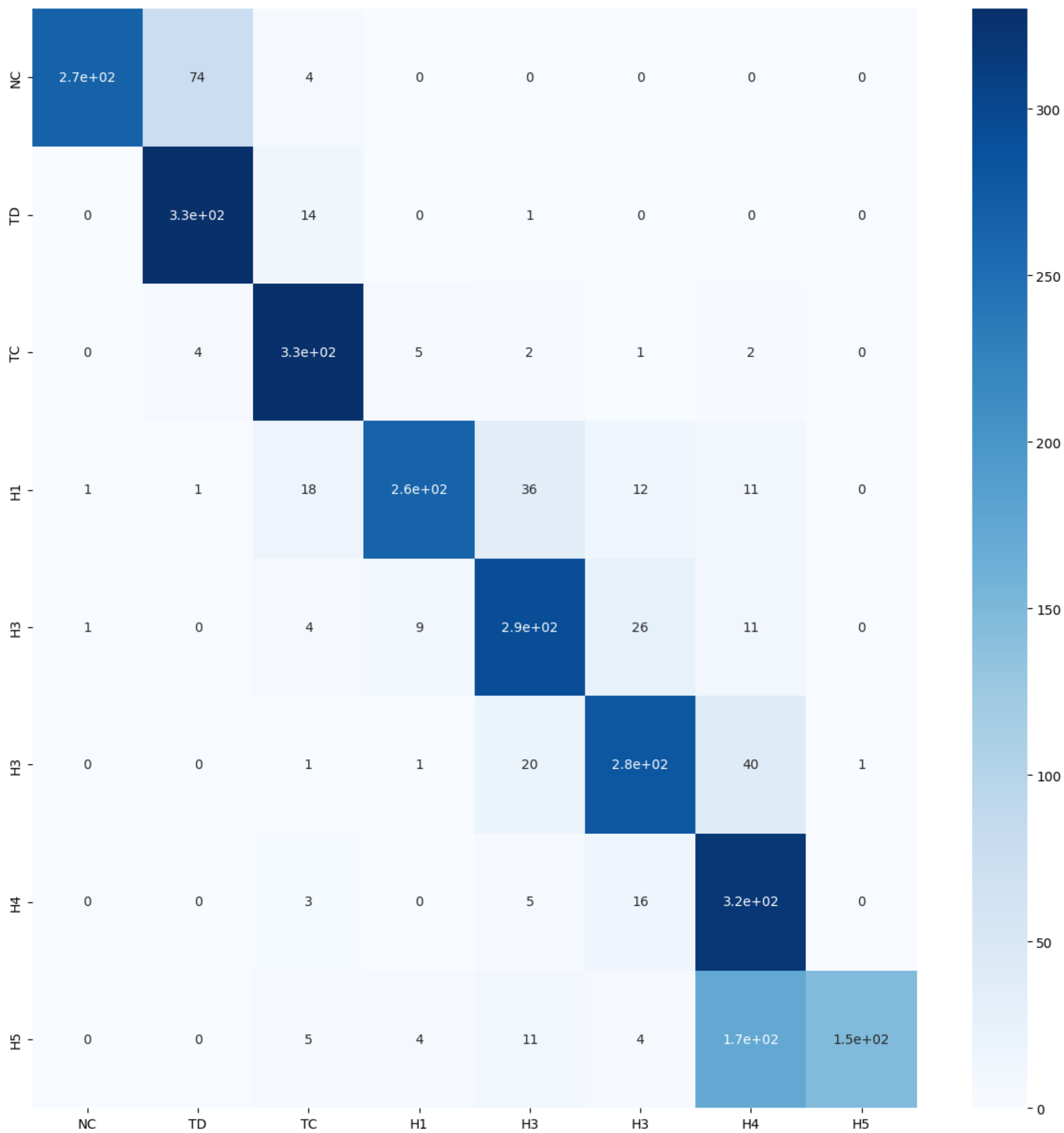
```
plt.show()
```



[18]:
```
#Plotting a heatmap using the confusion matrix
pred = model.predict(val)
p = np.argmax(pred, axis=1)
y_valid = np.argmax(val_labels, axis=1, out=None)
results = confusion_matrix(y_valid, p)
classes=['NC','TD','TC','H1','H3','H3','H4','H5']
df_cm = pd.DataFrame(results, index = [i for i in classes], columns = [i for i in classes])
plt.figure(figsize = (15,15))

sn.heatmap(df_cm, annot=True, cmap="Blues")
```

```
86/86 [==============================] - 3s 35ms/step
```

[18]:
```
<Axes: >
```



1.9 Modelo V2: Data Augmentation

Podemos observar que a precisão da validação é menor que a precisão do treinamento. Isso ocorre porque o modelo não está devidamente regularizado e as possíveis razões são: Poucos pontos de dados e Classes desequilibradas

A primeira coisa que é possível notar na contagem de categorias é que o número de imagens por categoria é muito desigual, com proporções de TC: H5 superiores a 1:20. Esse desequilíbrio pode enviesar a visão de nosso modelo CNN, pois prever errado na classe minoritária não afetaria muito o modelo, já que a contribuição da classe é inferior a 5% do conjunto de dados.

1.9.1 Data Augmentation

Assim, a técnica de data augmentation se torna necessária pra esse modelo. Basicamente, imagens com classes desiguais serão rotacionadas e invertidas para balancear a base de dados.

[19]:
```
def load_image(name, interpolation=cv2.INTER_AREA):
    img = cv2.imread(name, 1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    inter_area = cv2.resize(img, (256, 256), interpolation=interpolation)
    start_pt = np.random.randint(24, size=2)
    end_pt = start_pt + [232, 232]
    img = inter_area[start_pt[0]:end_pt[0], start_pt[1]:end_pt[1]]
    return img

def augmentation(name, category, filenames, labels, i):
    file_path = "Dataset/Aug/"
    (h, w) = (232, 232)
    center = (w / 2, h / 2)
    img = load_image(name, interpolation=cv2.INTER_LINEAR)
    augmentations = {0: lambda x: cv2.flip(x, 0), 7: lambda x: cv2.flip(x, 0)}
    images = [aug(img) for cat, aug in augmentations.items() if cat == category]

    for j, image in enumerate(images):
        filename = file_path + str(i + j) + '.jpeg'
        cv2.imwrite(filename, image)
        filenames.append(filename)
        labels.append(category)

    return i + len(images)
```

Vamos usar esse função ao carregar o dataset:

[20]:
```
filenames,labels = load_dataset(augment_fn = augmentation)
val_filenames , val_labels = make_test_set(filenames,labels,val=0.1)
test = 0.1
x_train, x_test, y_train, y_test = train_test_split(filenames, labels,␣
 ↪test_size=test, random_state=1)

y_train = tf.one_hot(y_train,depth=8)
y_test = tf.one_hot(y_test,depth=8)
val_labels = tf.one_hot(val_labels,depth=8)
train,test,val =␣
 ↪make_dataset((x_train,y_train,128),(x_test,y_test,32),(val_filenames,val_labels,32))
```

```
…
28574
[357, 357, 357, 357, 357, 357, 357, 357]
Counter({2: 7923, 3: 5326, 1: 3790, 4: 2921, 5: 2323, 6: 2165, 7: 739, 0: 531})
```

Treinando o modelo novamente:

[21]:

```
#Reset Graphs and Create Sequential model
K.clear_session()
model = Sequential()

#Convolution Layers
model.add(Conv2D(64, kernel_size=10,strides=3, activation='relu',␣
 ↪input_shape=(232,232,3)))
model.add(MaxPooling2D(pool_size=(3, 3),strides=2))
model.add(Conv2D(256, kernel_size=5,strides=1,activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3),strides=2))
model.add(Conv2D(288, kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=1))
model.add(Conv2D(272, kernel_size=3,strides=1,padding='same',activation='relu'))
model.add(Conv2D(256, kernel_size=3,strides=1,activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3),strides=2))
model.add(Dropout(0.5))
model.add(Flatten())

#Linear Layers
model.add(Dense(3584,activation='relu'))
model.add(Dense(2048,activation='relu'))
model.add(Dense(8, activation='softmax'))

model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 75, 75, 64)        19264

 max_pooling2d (MaxPooling2D  (None, 37, 37, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 33, 33, 256)       409856

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 16, 16, 288)       663840

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 288)      0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 272)       705296

 conv2d_4 (Conv2D)           (None, 13, 13, 256)       626944
```

```
 max_pooling2d_3 (MaxPooling  (None, 6, 6, 256)        0
 2D)

 dropout (Dropout)           (None, 6, 6, 256)         0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 3584)              33033728

 dense_1 (Dense)             (None, 2048)              7342080

 dense_2 (Dense)             (None, 8)                 16392

=================================================================
Total params: 42,817,400
Trainable params: 42,817,400
Non-trainable params: 0
_________________________________________________________________
```

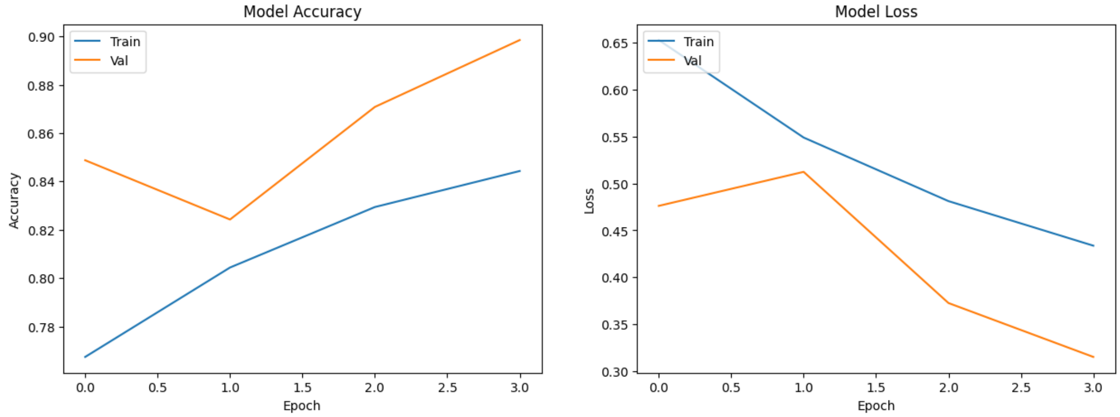
[22]:
```
# Include Top-2 Accuracy Metrics
top2_acc = functools.partial(keras.metrics.top_k_categorical_accuracy, k=2)
top2_acc.__name__ = 'top2_acc'

epochs = 4

model.load_weights("model.h5")

# Optimizer
sgd = keras.optimizers.legacy.SGD(learning_rate=0.001, decay=1e-6, momentum=0.9)

#Compile Model with Loss Function , Optimizer and Metrics
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=sgd,
              metrics=['accuracy',top2_acc])

# Train the Model
trained_model = model.fit(train,
          epochs=epochs,
          verbose=1,
          validation_data=val)

# Test Model Aganist Validation Set
score = model.evaluate(test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Epoch 1/4
181/181 [==============================] - 39s 212ms/step - loss: 0.6528 -
```

```
accuracy: 0.7674 - top2_acc: 0.9124 - val_loss: 0.4762 - val_accuracy: 0.8487 -
val_top2_acc: 0.9534
Epoch 2/4
181/181 [==============================] - 34s 185ms/step - loss: 0.5490 -
accuracy: 0.8044 - top2_acc: 0.9320 - val_loss: 0.5124 - val_accuracy: 0.8242 -
val_top2_acc: 0.9377
Epoch 3/4
181/181 [==============================] - 35s 191ms/step - loss: 0.4812 -
accuracy: 0.8294 - top2_acc: 0.9445 - val_loss: 0.3726 - val_accuracy: 0.8708 -
val_top2_acc: 0.9608
Epoch 4/4
181/181 [==============================] - 33s 183ms/step - loss: 0.4337 -
accuracy: 0.8443 - top2_acc: 0.9542 - val_loss: 0.3152 - val_accuracy: 0.8985 -
val_top2_acc: 0.9667
Test loss: 0.3379487991333008
Test accuracy: 0.8876360654830933
```

Visualizações:

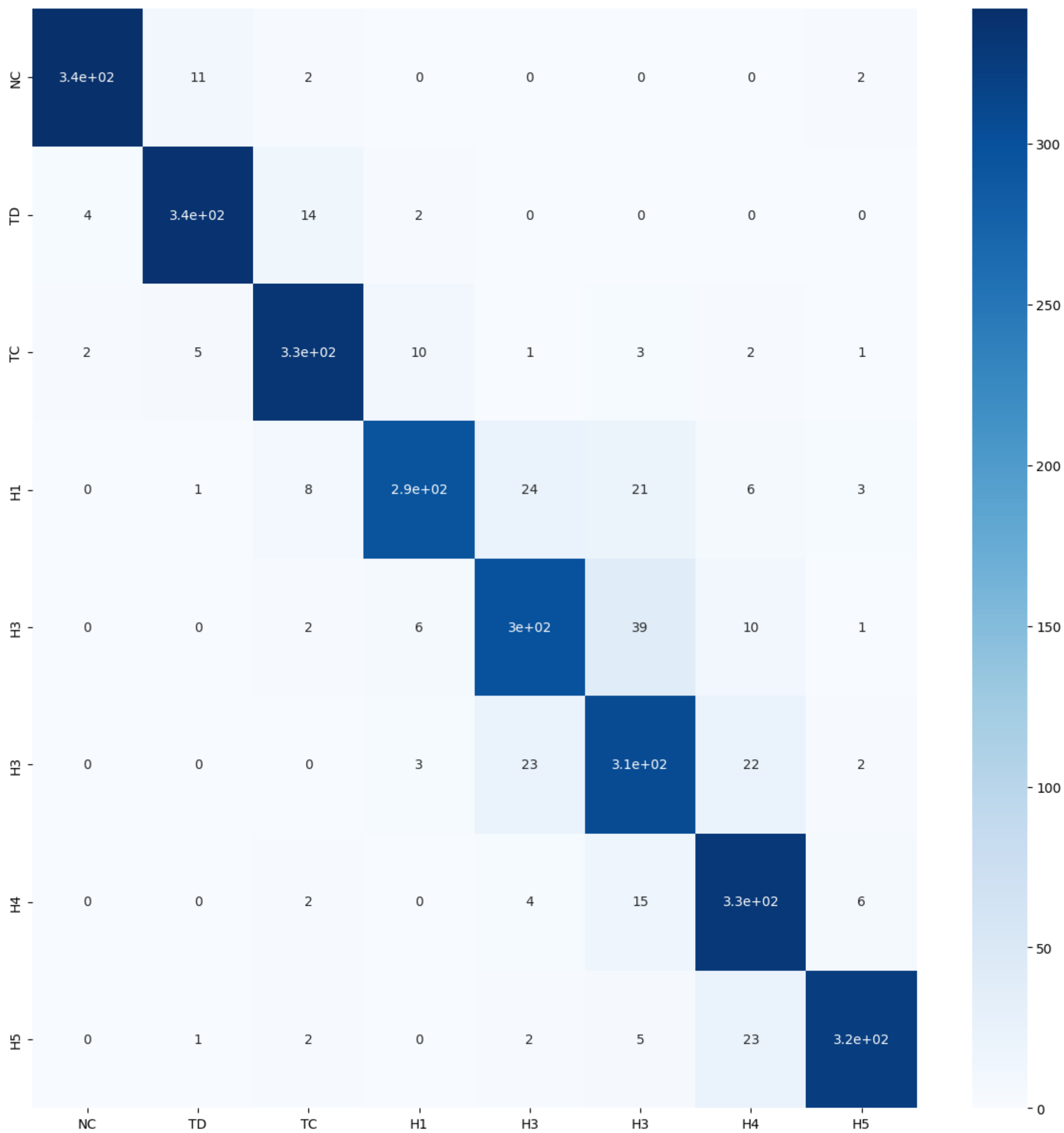
[23]:
```
f = plt.figure(figsize=(15,5))
ax = f.add_subplot(121)
ax.plot(trained_model.history['accuracy'])
ax.plot(trained_model.history['val_accuracy'])
ax.set_title('Model Accuracy')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.legend(['Train', 'Val'])

ax2 = f.add_subplot(122)
ax2.plot(trained_model.history['loss'])
ax2.plot(trained_model.history['val_loss'])
ax2.set_title('Model Loss')
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend(['Train', 'Val'],loc= 'upper left')

plt.show()
```



[24]:
```
pred = model.predict(val)
p = np.argmax(pred, axis=1)
y_valid = np.argmax(val_labels, axis=1, out=None)
results = confusion_matrix(y_valid, p)
classes=['NC','TD','TC','H1','H3','H3','H4','H5']
df_cm = pd.DataFrame(results, index = [i for i in classes], columns = [i for i␣
↪in classes])
plt.figure(figsize = (15,15))

sn.heatmap(df_cm, annot=True, cmap="Blues")
```

```
90/90 [==============================] - 4s 39ms/step
```

[24]: <Axes: >



O modelo obteve uma melhor acurácia na validação em relação ao treinamento. Portanto, vamos salvar o modelo para futuras modificações.

[25]:
```
model.save('model_v2.h5')
```

1.10 Modelo V3: Otimização de Hiperparâmetros

[26]:
```
filenames,labels = load_dataset(augment_fn = augmentation)
val_filenames , val_labels = make_test_set(filenames,labels,val=0.1)
test = 0.1
```

```
x_train, x_test, y_train, y_test = train_test_split(filenames, labels,␣
 ↪test_size=test, random_state=1)
y_train = tf.one_hot(y_train,depth=8)
y_test = tf.one_hot(y_test,depth=8)
val_labels = tf.one_hot(val_labels,depth=8)
```

…
28574
[357, 357, 357, 357, 357, 357, 357, 357]
Counter({2: 7923, 3: 5326, 1: 3790, 4: 2921, 5: 2323, 6: 2165, 7: 739, 0: 531})

[27]:
```
def build_model():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=10, strides=3, activation='relu',␣
 ↪input_shape=(232, 232, 3)))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
    model.add(Conv2D(256, kernel_size=5, strides=1, activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
    model.add(Conv2D(288, kernel_size=3, strides=1, padding='same',␣
 ↪activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=1))
    model.add(Conv2D(272, kernel_size=3, strides=1, padding='same',␣
 ↪activation='relu'))
    model.add(Conv2D(256, kernel_size=3, strides=1, activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(3584, activation='relu'))
    model.add(Dense(2048, activation='relu'))
    model.add(Dense(8, activation='softmax'))

    return model
```

Assim foi realizado um grid search simples nos hiperparâmetros: **batch size**, **epoch* e** função de otimização**. O espaço de busca é reduzido devido ao tempo de treinamento do modelo:

[28]:
```
def hyperparameter_tuning():
    batch_sizes = [32, 64]
    epoch_list = [12, 24]
    optimizers = [
        keras.optimizers.legacy.SGD(learning_rate=0.001, decay=1e-6, momentum=0.
 ↪9),
        keras.optimizers.legacy.Adam(learning_rate=0.001)
    ]

    best_accuracy = 0.0
    best_hyperparameters = None
    best_model = None
```

```
    for batch_size in batch_sizes:
        for epochs in epoch_list:
            for optimizer in optimizers:
                train, test, val = make_dataset((x_train, y_train, batch_size),␣
↪(x_test, y_test, 32), (val_filenames, val_labels, 32))
                K.clear_session()
                model = build_model()
                model.compile(loss=keras.losses.categorical_crossentropy,␣
↪optimizer=optimizer, metrics=['accuracy', top2_acc])

                trained_model = model.fit(train, epochs=epochs, verbose=1,␣
↪validation_data=val)
                score = model.evaluate(test, verbose=0)

                if score[1] > best_accuracy:
                    best_accuracy = score[1]
                    best_hyperparameters = (batch_size, epochs, optimizer)
                    best_model = model

    return best_hyperparameters, best_model
```

[29]:
```
best_hyperparameters, best_model = hyperparameter_tuning()
print("Best hyperparameters:", best_hyperparameters)
```

```
/usr/local/lib/python3.10/dist-
packages/keras/optimizers/legacy/gradient_descent.py:114: UserWarning: The `lr`
argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)

Epoch 1/12
724/724 [==============================] - 45s 61ms/step - loss: 1.6973 -
accuracy: 0.3421 - top2_acc: 0.5753 - val_loss: 1.8689 - val_accuracy: 0.2812 -
val_top2_acc: 0.4790
Epoch 2/12
724/724 [==============================] - 43s 59ms/step - loss: 1.3738 -
accuracy: 0.4451 - top2_acc: 0.6987 - val_loss: 1.3756 - val_accuracy: 0.4356 -
val_top2_acc: 0.6786
Epoch 3/12
724/724 [==============================] - 41s 56ms/step - loss: 1.1726 -
accuracy: 0.5321 - top2_acc: 0.7689 - val_loss: 1.0240 - val_accuracy: 0.5963 -
val_top2_acc: 0.8036
Epoch 4/12
724/724 [==============================] - 39s 53ms/step - loss: 0.9866 -
accuracy: 0.6153 - top2_acc: 0.8285 - val_loss: 0.8328 - val_accuracy: 0.6702 -
```

```
val_top2_acc: 0.8617
Epoch 5/12
724/724 [==============================] - 41s 56ms/step - loss: 0.8024 -
accuracy: 0.6919 - top2_acc: 0.8773 - val_loss: 0.5759 - val_accuracy: 0.7994 -
val_top2_acc: 0.9261
Epoch 6/12
724/724 [==============================] - 37s 52ms/step - loss: 0.6398 -
accuracy: 0.7617 - top2_acc: 0.9162 - val_loss: 0.4695 - val_accuracy: 0.8270 -
val_top2_acc: 0.9499
Epoch 7/12
724/724 [==============================] - 40s 56ms/step - loss: 0.4900 -
accuracy: 0.8199 - top2_acc: 0.9470 - val_loss: 0.3488 - val_accuracy: 0.8827 -
val_top2_acc: 0.9636
Epoch 8/12
724/724 [==============================] - 38s 52ms/step - loss: 0.3871 -
accuracy: 0.8579 - top2_acc: 0.9621 - val_loss: 0.2376 - val_accuracy: 0.9160 -
val_top2_acc: 0.9790
Epoch 9/12
724/724 [==============================] - 41s 57ms/step - loss: 0.3064 -
accuracy: 0.8907 - top2_acc: 0.9749 - val_loss: 0.2127 - val_accuracy: 0.9237 -
val_top2_acc: 0.9832
Epoch 10/12
724/724 [==============================] - 41s 57ms/step - loss: 0.2430 -
accuracy: 0.9129 - top2_acc: 0.9819 - val_loss: 0.1631 - val_accuracy: 0.9405 -
val_top2_acc: 0.9926
Epoch 11/12
724/724 [==============================] - 41s 56ms/step - loss: 0.1996 -
accuracy: 0.9269 - top2_acc: 0.9879 - val_loss: 0.1643 - val_accuracy: 0.9391 -
val_top2_acc: 0.9916
Epoch 12/12
724/724 [==============================] - 41s 56ms/step - loss: 0.1744 -
accuracy: 0.9393 - top2_acc: 0.9908 - val_loss: 0.1208 - val_accuracy: 0.9555 -
val_top2_acc: 0.9933
Epoch 1/12
724/724 [==============================] - 41s 55ms/step - loss: 1.8485 -
accuracy: 0.3065 - top2_acc: 0.5138 - val_loss: 2.4090 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 2/12
724/724 [==============================] - 40s 55ms/step - loss: 1.8313 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3682 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 3/12
724/724 [==============================] - 38s 53ms/step - loss: 1.8309 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4587 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 4/12
724/724 [==============================] - 41s 56ms/step - loss: 1.8299 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3477 - val_accuracy: 0.1250 -
```

```
val_top2_acc: 0.2500
Epoch 5/12
724/724 [==============================] - 45s 63ms/step - loss: 1.8298 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3781 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 6/12
724/724 [==============================] - 38s 52ms/step - loss: 1.8297 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3962 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 7/12
724/724 [==============================] - 38s 53ms/step - loss: 1.8298 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4146 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 8/12
724/724 [==============================] - 38s 53ms/step - loss: 1.8294 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3170 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 9/12
724/724 [==============================] - 37s 51ms/step - loss: 1.8295 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3739 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 10/12
724/724 [==============================] - 40s 56ms/step - loss: 1.8293 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3356 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 11/12
724/724 [==============================] - 40s 55ms/step - loss: 1.8292 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3768 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 12/12
724/724 [==============================] - 39s 53ms/step - loss: 1.8292 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4026 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 1/24
724/724 [==============================] - 40s 54ms/step - loss: 1.6983 -
accuracy: 0.3392 - top2_acc: 0.5713 - val_loss: 1.7694 - val_accuracy: 0.2990 -
val_top2_acc: 0.4709
Epoch 2/24
724/724 [==============================] - 38s 53ms/step - loss: 1.3886 -
accuracy: 0.4336 - top2_acc: 0.6875 - val_loss: 1.3128 - val_accuracy: 0.4772 -
val_top2_acc: 0.6996
Epoch 3/24
724/724 [==============================] - 41s 57ms/step - loss: 1.1884 -
accuracy: 0.5220 - top2_acc: 0.7650 - val_loss: 1.0226 - val_accuracy: 0.6145 -
val_top2_acc: 0.8235
Epoch 4/24
724/724 [==============================] - 40s 55ms/step - loss: 1.0095 -
accuracy: 0.6002 - top2_acc: 0.8183 - val_loss: 0.8106 - val_accuracy: 0.6999 -
```

```
val_top2_acc: 0.8743
Epoch 5/24
724/724 [==============================] - 38s 53ms/step - loss: 0.8287 -
accuracy: 0.6822 - top2_acc: 0.8716 - val_loss: 0.6673 - val_accuracy: 0.7416 -
val_top2_acc: 0.9044
Epoch 6/24
724/724 [==============================] - 40s 55ms/step - loss: 0.6622 -
accuracy: 0.7499 - top2_acc: 0.9107 - val_loss: 0.4870 - val_accuracy: 0.8179 -
val_top2_acc: 0.9394
Epoch 7/24
724/724 [==============================] - 40s 55ms/step - loss: 0.5239 -
accuracy: 0.8038 - top2_acc: 0.9397 - val_loss: 0.3630 - val_accuracy: 0.8757 -
val_top2_acc: 0.9650
Epoch 8/24
724/724 [==============================] - 40s 55ms/step - loss: 0.4128 -
accuracy: 0.8506 - top2_acc: 0.9594 - val_loss: 0.2796 - val_accuracy: 0.8999 -
val_top2_acc: 0.9737
Epoch 9/24
724/724 [==============================] - 40s 56ms/step - loss: 0.3263 -
accuracy: 0.8818 - top2_acc: 0.9723 - val_loss: 0.2141 - val_accuracy: 0.9219 -
val_top2_acc: 0.9821
Epoch 10/24
724/724 [==============================] - 41s 56ms/step - loss: 0.2649 -
accuracy: 0.9056 - top2_acc: 0.9810 - val_loss: 0.1927 - val_accuracy: 0.9293 -
val_top2_acc: 0.9863
Epoch 11/24
724/724 [==============================] - 39s 53ms/step - loss: 0.2152 -
accuracy: 0.9223 - top2_acc: 0.9860 - val_loss: 0.1719 - val_accuracy: 0.9415 -
val_top2_acc: 0.9891
Epoch 12/24
724/724 [==============================] - 39s 53ms/step - loss: 0.1819 -
accuracy: 0.9365 - top2_acc: 0.9895 - val_loss: 0.1221 - val_accuracy: 0.9562 -
val_top2_acc: 0.9954
Epoch 13/24
724/724 [==============================] - 41s 57ms/step - loss: 0.1536 -
accuracy: 0.9466 - top2_acc: 0.9927 - val_loss: 0.1324 - val_accuracy: 0.9573 -
val_top2_acc: 0.9961
Epoch 14/24
724/724 [==============================] - 40s 56ms/step - loss: 0.1291 -
accuracy: 0.9543 - top2_acc: 0.9948 - val_loss: 0.1090 - val_accuracy: 0.9597 -
val_top2_acc: 0.9961
Epoch 15/24
724/724 [==============================] - 40s 55ms/step - loss: 0.1198 -
accuracy: 0.9575 - top2_acc: 0.9952 - val_loss: 0.0963 - val_accuracy: 0.9681 -
val_top2_acc: 0.9958
Epoch 16/24
724/724 [==============================] - 40s 55ms/step - loss: 0.1065 -
accuracy: 0.9618 - top2_acc: 0.9971 - val_loss: 0.0829 - val_accuracy: 0.9734 -
```

```
val_top2_acc: 0.9972
Epoch 17/24
724/724 [==============================] - 40s 55ms/step - loss: 0.0948 -
accuracy: 0.9651 - top2_acc: 0.9972 - val_loss: 0.0873 - val_accuracy: 0.9695 -
val_top2_acc: 0.9968
Epoch 18/24
724/724 [==============================] - 41s 56ms/step - loss: 0.0840 -
accuracy: 0.9708 - top2_acc: 0.9976 - val_loss: 0.0857 - val_accuracy: 0.9737 -
val_top2_acc: 0.9979
Epoch 19/24
724/724 [==============================] - 41s 57ms/step - loss: 0.0774 -
accuracy: 0.9736 - top2_acc: 0.9985 - val_loss: 0.0790 - val_accuracy: 0.9737 -
val_top2_acc: 0.9986
Epoch 20/24
724/724 [==============================] - 41s 57ms/step - loss: 0.0702 -
accuracy: 0.9749 - top2_acc: 0.9987 - val_loss: 0.0899 - val_accuracy: 0.9727 -
val_top2_acc: 0.9975
Epoch 21/24
724/724 [==============================] - 39s 54ms/step - loss: 0.0702 -
accuracy: 0.9750 - top2_acc: 0.9988 - val_loss: 0.0822 - val_accuracy: 0.9727 -
val_top2_acc: 0.9986
Epoch 22/24
724/724 [==============================] - 41s 57ms/step - loss: 0.0617 -
accuracy: 0.9793 - top2_acc: 0.9987 - val_loss: 0.0773 - val_accuracy: 0.9730 -
val_top2_acc: 0.9982
Epoch 23/24
724/724 [==============================] - 39s 54ms/step - loss: 0.0594 -
accuracy: 0.9797 - top2_acc: 0.9990 - val_loss: 0.0830 - val_accuracy: 0.9741 -
val_top2_acc: 0.9982
Epoch 24/24
724/724 [==============================] - 40s 55ms/step - loss: 0.0588 -
accuracy: 0.9803 - top2_acc: 0.9992 - val_loss: 0.0691 - val_accuracy: 0.9776 -
val_top2_acc: 0.9986
Epoch 1/24
724/724 [==============================] - 40s 54ms/step - loss: 9.0205 -
accuracy: 0.3033 - top2_acc: 0.5117 - val_loss: 2.4401 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 2/24
724/724 [==============================] - 41s 57ms/step - loss: 1.8297 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3648 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 3/24
724/724 [==============================] - 40s 56ms/step - loss: 1.8292 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3846 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 4/24
724/724 [==============================] - 38s 53ms/step - loss: 1.8291 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3970 - val_accuracy: 0.1250 -
```

```
val_top2_acc: 0.2500
Epoch 5/24
724/724 [==============================] - 38s 52ms/step - loss: 1.8289 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3491 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 6/24
724/724 [==============================] - 40s 55ms/step - loss: 1.8289 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4251 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 7/24
724/724 [==============================] - 40s 55ms/step - loss: 1.8288 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3755 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 8/24
724/724 [==============================] - 40s 55ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4052 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 9/24
724/724 [==============================] - 40s 55ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3797 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 10/24
724/724 [==============================] - 40s 55ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4116 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 11/24
724/724 [==============================] - 38s 53ms/step - loss: 1.8284 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4043 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 12/24
724/724 [==============================] - 39s 54ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3647 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 13/24
724/724 [==============================] - 38s 53ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3878 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 14/24
724/724 [==============================] - 40s 55ms/step - loss: 1.8283 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4257 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 15/24
724/724 [==============================] - 39s 54ms/step - loss: 1.8284 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3885 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 16/24
724/724 [==============================] - 39s 54ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4278 - val_accuracy: 0.1250 -
```

```
val_top2_acc: 0.2500
Epoch 17/24
724/724 [==============================] - 37s 52ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4085 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 18/24
724/724 [==============================] - 39s 54ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3788 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 19/24
724/724 [==============================] - 37s 52ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3969 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 20/24
724/724 [==============================] - 39s 54ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3702 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 21/24
724/724 [==============================] - 38s 52ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3770 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 22/24
724/724 [==============================] - 37s 51ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3910 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 23/24
724/724 [==============================] - 39s 53ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3875 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 24/24
724/724 [==============================] - 39s 53ms/step - loss: 1.8284 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4085 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 1/12
362/362 [==============================] - 38s 98ms/step - loss: 1.8330 -
accuracy: 0.3020 - top2_acc: 0.5138 - val_loss: 2.3451 - val_accuracy: 0.1429 -
val_top2_acc: 0.3249
Epoch 2/12
362/362 [==============================] - 35s 97ms/step - loss: 1.5608 -
accuracy: 0.3767 - top2_acc: 0.6320 - val_loss: 1.7164 - val_accuracy: 0.3193 -
val_top2_acc: 0.5098
Epoch 3/12
362/362 [==============================] - 36s 98ms/step - loss: 1.4099 -
accuracy: 0.4269 - top2_acc: 0.6830 - val_loss: 1.5249 - val_accuracy: 0.3953 -
val_top2_acc: 0.6022
Epoch 4/12
362/362 [==============================] - 34s 92ms/step - loss: 1.2843 -
accuracy: 0.4822 - top2_acc: 0.7361 - val_loss: 1.2610 - val_accuracy: 0.5070 -
```

```
val_top2_acc: 0.7283
Epoch 5/12
362/362 [==============================] - 34s 94ms/step - loss: 1.1540 -
accuracy: 0.5412 - top2_acc: 0.7761 - val_loss: 1.1153 - val_accuracy: 0.5721 -
val_top2_acc: 0.7756
Epoch 6/12
362/362 [==============================] - 36s 98ms/step - loss: 1.0358 -
accuracy: 0.5947 - top2_acc: 0.8148 - val_loss: 0.9957 - val_accuracy: 0.6201 -
val_top2_acc: 0.8298
Epoch 7/12
362/362 [==============================] - 35s 97ms/step - loss: 0.9126 -
accuracy: 0.6463 - top2_acc: 0.8501 - val_loss: 0.7405 - val_accuracy: 0.7377 -
val_top2_acc: 0.8992
Epoch 8/12
362/362 [==============================] - 36s 98ms/step - loss: 0.7957 -
accuracy: 0.6970 - top2_acc: 0.8797 - val_loss: 0.6403 - val_accuracy: 0.7598 -
val_top2_acc: 0.9135
Epoch 9/12
362/362 [==============================] - 33s 91ms/step - loss: 0.6801 -
accuracy: 0.7432 - top2_acc: 0.9094 - val_loss: 0.5718 - val_accuracy: 0.7868 -
val_top2_acc: 0.9300
Epoch 10/12
362/362 [==============================] - 35s 98ms/step - loss: 0.5785 -
accuracy: 0.7826 - top2_acc: 0.9294 - val_loss: 0.4804 - val_accuracy: 0.8249 -
val_top2_acc: 0.9349
Epoch 11/12
362/362 [==============================] - 33s 91ms/step - loss: 0.4910 -
accuracy: 0.8197 - top2_acc: 0.9446 - val_loss: 0.4167 - val_accuracy: 0.8379 -
val_top2_acc: 0.9573
Epoch 12/12
362/362 [==============================] - 35s 95ms/step - loss: 0.3983 -
accuracy: 0.8551 - top2_acc: 0.9601 - val_loss: 0.2660 - val_accuracy: 0.9051 -
val_top2_acc: 0.9681
Epoch 1/12
362/362 [==============================] - 38s 101ms/step - loss: 5.7874 -
accuracy: 0.3033 - top2_acc: 0.5067 - val_loss: 2.4107 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 2/12
362/362 [==============================] - 36s 98ms/step - loss: 1.8301 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4001 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 3/12
362/362 [==============================] - 34s 94ms/step - loss: 1.8290 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3568 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 4/12
362/362 [==============================] - 34s 95ms/step - loss: 1.8290 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3462 - val_accuracy: 0.1250 -
```

```
val_top2_acc: 0.2500
Epoch 5/12
362/362 [==============================] - 35s 97ms/step - loss: 1.8295 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3944 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 6/12
362/362 [==============================] - 37s 102ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4026 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 7/12
362/362 [==============================] - 37s 101ms/step - loss: 1.8289 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3999 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 8/12
362/362 [==============================] - 33s 91ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3447 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 9/12
362/362 [==============================] - 36s 100ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4175 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 10/12
362/362 [==============================] - 33s 92ms/step - loss: 1.8284 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3894 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 11/12
362/362 [==============================] - 37s 103ms/step - loss: 1.8288 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3989 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 12/12
362/362 [==============================] - 36s 99ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3859 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 1/24
362/362 [==============================] - 37s 101ms/step - loss: 1.8278 -
accuracy: 0.3095 - top2_acc: 0.5178 - val_loss: 2.3028 - val_accuracy: 0.1744 -
val_top2_acc: 0.3022
Epoch 2/24
362/362 [==============================] - 36s 99ms/step - loss: 1.5537 -
accuracy: 0.3786 - top2_acc: 0.6329 - val_loss: 1.6776 - val_accuracy: 0.3365 -
val_top2_acc: 0.5322
Epoch 3/24
362/362 [==============================] - 33s 92ms/step - loss: 1.3831 -
accuracy: 0.4387 - top2_acc: 0.6947 - val_loss: 1.4677 - val_accuracy: 0.4093 -
val_top2_acc: 0.6271
Epoch 4/24
362/362 [==============================] - 33s 91ms/step - loss: 1.2503 -
accuracy: 0.5008 - top2_acc: 0.7449 - val_loss: 1.2214 - val_accuracy: 0.5217 -
```

```
val_top2_acc: 0.7430
Epoch 5/24
362/362 [==============================] - 33s 90ms/step - loss: 1.1153 -
accuracy: 0.5579 - top2_acc: 0.7912 - val_loss: 1.1158 - val_accuracy: 0.5567 -
val_top2_acc: 0.7882
Epoch 6/24
362/362 [==============================] - 34s 95ms/step - loss: 0.9847 -
accuracy: 0.6135 - top2_acc: 0.8313 - val_loss: 0.8164 - val_accuracy: 0.6919 -
val_top2_acc: 0.8803
Epoch 7/24
362/362 [==============================] - 36s 99ms/step - loss: 0.8616 -
accuracy: 0.6694 - top2_acc: 0.8656 - val_loss: 0.7353 - val_accuracy: 0.7349 -
val_top2_acc: 0.8911
Epoch 8/24
362/362 [==============================] - 37s 101ms/step - loss: 0.7424 -
accuracy: 0.7210 - top2_acc: 0.8923 - val_loss: 0.5266 - val_accuracy: 0.8137 -
val_top2_acc: 0.9314
Epoch 9/24
362/362 [==============================] - 36s 100ms/step - loss: 0.6348 -
accuracy: 0.7652 - top2_acc: 0.9173 - val_loss: 0.4767 - val_accuracy: 0.8428 -
val_top2_acc: 0.9408
Epoch 10/24
362/362 [==============================] - 37s 101ms/step - loss: 0.5351 -
accuracy: 0.8020 - top2_acc: 0.9364 - val_loss: 0.3824 - val_accuracy: 0.8704 -
val_top2_acc: 0.9650
Epoch 11/24
362/362 [==============================] - 36s 99ms/step - loss: 0.4553 -
accuracy: 0.8326 - top2_acc: 0.9503 - val_loss: 0.2950 - val_accuracy: 0.8981 -
val_top2_acc: 0.9751
Epoch 12/24
362/362 [==============================] - 34s 93ms/step - loss: 0.3794 -
accuracy: 0.8616 - top2_acc: 0.9629 - val_loss: 0.2704 - val_accuracy: 0.9086 -
val_top2_acc: 0.9772
Epoch 13/24
362/362 [==============================] - 35s 97ms/step - loss: 0.3274 -
accuracy: 0.8817 - top2_acc: 0.9723 - val_loss: 0.2126 - val_accuracy: 0.9286 -
val_top2_acc: 0.9870
Epoch 14/24
362/362 [==============================] - 32s 88ms/step - loss: 0.2815 -
accuracy: 0.9000 - top2_acc: 0.9773 - val_loss: 0.2023 - val_accuracy: 0.9331 -
val_top2_acc: 0.9821
Epoch 15/24
362/362 [==============================] - 36s 99ms/step - loss: 0.2417 -
accuracy: 0.9143 - top2_acc: 0.9818 - val_loss: 0.1495 - val_accuracy: 0.9503 -
val_top2_acc: 0.9898
Epoch 16/24
362/362 [==============================] - 36s 98ms/step - loss: 0.2124 -
accuracy: 0.9247 - top2_acc: 0.9863 - val_loss: 0.1480 - val_accuracy: 0.9496 -
```

```
val_top2_acc: 0.9898
Epoch 17/24
362/362 [==============================] - 36s 99ms/step - loss: 0.1817 -
accuracy: 0.9348 - top2_acc: 0.9896 - val_loss: 0.1230 - val_accuracy: 0.9576 -
val_top2_acc: 0.9905
Epoch 18/24
362/362 [==============================] - 34s 94ms/step - loss: 0.1631 -
accuracy: 0.9415 - top2_acc: 0.9908 - val_loss: 0.1595 - val_accuracy: 0.9478 -
val_top2_acc: 0.9888
Epoch 19/24
362/362 [==============================] - 34s 95ms/step - loss: 0.1505 -
accuracy: 0.9453 - top2_acc: 0.9921 - val_loss: 0.1251 - val_accuracy: 0.9576 -
val_top2_acc: 0.9902
Epoch 20/24
362/362 [==============================] - 36s 99ms/step - loss: 0.1354 -
accuracy: 0.9529 - top2_acc: 0.9937 - val_loss: 0.1003 - val_accuracy: 0.9629 -
val_top2_acc: 0.9958
Epoch 21/24
362/362 [==============================] - 32s 89ms/step - loss: 0.1235 -
accuracy: 0.9564 - top2_acc: 0.9949 - val_loss: 0.0997 - val_accuracy: 0.9639 -
val_top2_acc: 0.9937
Epoch 22/24
362/362 [==============================] - 36s 100ms/step - loss: 0.1168 -
accuracy: 0.9591 - top2_acc: 0.9960 - val_loss: 0.0924 - val_accuracy: 0.9681 -
val_top2_acc: 0.9951
Epoch 23/24
362/362 [==============================] - 33s 90ms/step - loss: 0.1006 -
accuracy: 0.9633 - top2_acc: 0.9965 - val_loss: 0.1042 - val_accuracy: 0.9653 -
val_top2_acc: 0.9947
Epoch 24/24
362/362 [==============================] - 34s 94ms/step - loss: 0.0960 -
accuracy: 0.9661 - top2_acc: 0.9968 - val_loss: 0.0953 - val_accuracy: 0.9671 -
val_top2_acc: 0.9961
Epoch 1/24
362/362 [==============================] - 37s 99ms/step - loss: 13.6518 -
accuracy: 0.3028 - top2_acc: 0.5115 - val_loss: 2.3744 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 2/24
362/362 [==============================] - 33s 92ms/step - loss: 1.8293 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3570 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 3/24
362/362 [==============================] - 36s 100ms/step - loss: 1.8294 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4091 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 4/24
362/362 [==============================] - 34s 93ms/step - loss: 1.8290 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3343 - val_accuracy: 0.1250 -
```

```
val_top2_acc: 0.2500
Epoch 5/24
362/362 [==============================] - 35s 97ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3682 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 6/24
362/362 [==============================] - 35s 96ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3700 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 7/24
362/362 [==============================] - 32s 89ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4254 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 8/24
362/362 [==============================] - 31s 86ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3980 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 9/24
362/362 [==============================] - 35s 96ms/step - loss: 1.8288 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4144 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 10/24
362/362 [==============================] - 32s 88ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4263 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 11/24
362/362 [==============================] - 34s 92ms/step - loss: 1.8284 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3999 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 12/24
362/362 [==============================] - 34s 95ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3891 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 13/24
362/362 [==============================] - 31s 85ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3734 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 14/24
362/362 [==============================] - 32s 87ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3995 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 15/24
362/362 [==============================] - 34s 93ms/step - loss: 1.8287 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3991 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 16/24
362/362 [==============================] - 32s 89ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3922 - val_accuracy: 0.1250 -
```

```
val_top2_acc: 0.2500
Epoch 17/24
362/362 [==============================] - 32s 88ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3875 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 18/24
362/362 [==============================] - 32s 88ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4182 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 19/24
362/362 [==============================] - 33s 90ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4093 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 20/24
362/362 [==============================] - 31s 86ms/step - loss: 1.8286 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4181 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 21/24
362/362 [==============================] - 32s 88ms/step - loss: 1.8283 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4227 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 22/24
362/362 [==============================] - 34s 93ms/step - loss: 1.8283 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4123 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 23/24
362/362 [==============================] - 32s 89ms/step - loss: 1.8283 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.4013 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Epoch 24/24
362/362 [==============================] - 33s 90ms/step - loss: 1.8285 -
accuracy: 0.3073 - top2_acc: 0.5145 - val_loss: 2.3718 - val_accuracy: 0.1250 -
val_top2_acc: 0.2500
Best hyperparameters: (32, 24, <keras.optimizers.legacy.gradient_descent.SGD
object at 0x783e080f0220>)
```

Portanto os melhores hiperparâmetros encontrados foram:

- Batch Size = 32
- Epochs = 24
- Função de otimização = Stocastic Gradient Descend(keras.optimizers.legacy.SGD(learning_rate=0.001, decay=1e-6, momentum=0.9)_)

[31]:
```
best_batch_size, best_epochs, best_optimizer = best_hyperparameters
train, test, val = make_dataset((x_train, y_train, best_batch_size), (x_test,␣
 ↪y_test, 32), (val_filenames, val_labels, 32))
best_model.compile(loss=keras.losses.categorical_crossentropy,␣
 ↪optimizer=best_optimizer, metrics=['accuracy', top2_acc])
```

```
best_history = best_model.fit(train, epochs=best_epochs, verbose=1,␣
 ↪validation_data=val)
```

```
Epoch 1/24
724/724 [==============================] - 48s 65ms/step - loss: 0.0484 -
accuracy: 0.9834 - top2_acc: 0.9996 - val_loss: 0.0819 - val_accuracy: 0.9758 -
val_top2_acc: 0.9982
Epoch 2/24
724/724 [==============================] - 39s 54ms/step - loss: 0.0517 -
accuracy: 0.9837 - top2_acc: 0.9990 - val_loss: 0.0860 - val_accuracy: 0.9734 -
val_top2_acc: 0.9972
Epoch 3/24
724/724 [==============================] - 40s 55ms/step - loss: 0.0467 -
accuracy: 0.9849 - top2_acc: 0.9989 - val_loss: 0.0706 - val_accuracy: 0.9793 -
val_top2_acc: 0.9989
Epoch 4/24
724/724 [==============================] - 40s 55ms/step - loss: 0.0477 -
accuracy: 0.9833 - top2_acc: 0.9993 - val_loss: 0.0752 - val_accuracy: 0.9800 -
val_top2_acc: 0.9982
Epoch 5/24
724/724 [==============================] - 39s 53ms/step - loss: 0.0423 -
accuracy: 0.9862 - top2_acc: 0.9994 - val_loss: 0.0761 - val_accuracy: 0.9776 -
val_top2_acc: 0.9982
Epoch 6/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0394 -
accuracy: 0.9858 - top2_acc: 0.9994 - val_loss: 0.0694 - val_accuracy: 0.9793 -
val_top2_acc: 0.9989
Epoch 7/24
724/724 [==============================] - 38s 53ms/step - loss: 0.0384 -
accuracy: 0.9869 - top2_acc: 0.9995 - val_loss: 0.0675 - val_accuracy: 0.9786 -
val_top2_acc: 0.9989
Epoch 8/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0380 -
accuracy: 0.9866 - top2_acc: 0.9996 - val_loss: 0.0710 - val_accuracy: 0.9779 -
val_top2_acc: 0.9986
Epoch 9/24
724/724 [==============================] - 37s 50ms/step - loss: 0.0356 -
accuracy: 0.9880 - top2_acc: 0.9997 - val_loss: 0.0778 - val_accuracy: 0.9779 -
val_top2_acc: 0.9989
Epoch 10/24
724/724 [==============================] - 37s 52ms/step - loss: 0.0364 -
accuracy: 0.9875 - top2_acc: 0.9996 - val_loss: 0.0831 - val_accuracy: 0.9762 -
val_top2_acc: 0.9986
Epoch 11/24
724/724 [==============================] - 36s 50ms/step - loss: 0.0344 -
accuracy: 0.9877 - top2_acc: 0.9997 - val_loss: 0.0784 - val_accuracy: 0.9751 -
val_top2_acc: 0.9979
Epoch 12/24
```

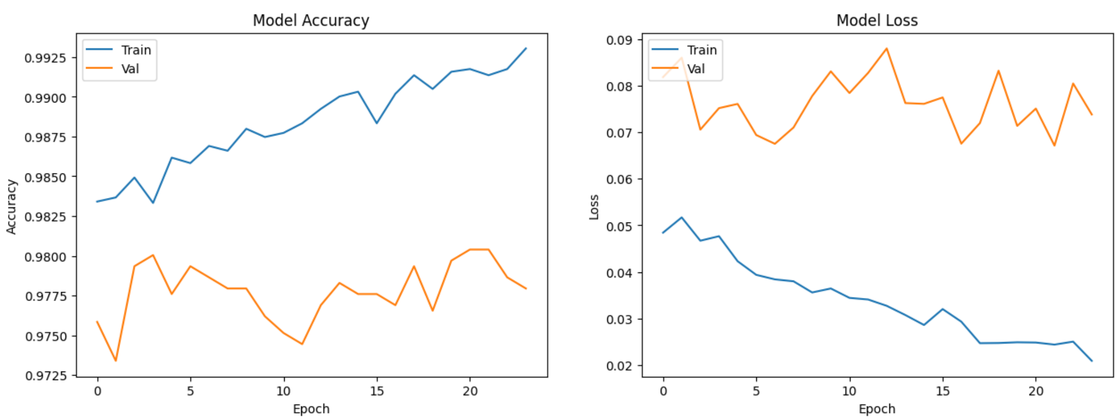
```
724/724 [==============================] - 37s 51ms/step - loss: 0.0341 -
accuracy: 0.9883 - top2_acc: 0.9997 - val_loss: 0.0828 - val_accuracy: 0.9744 -
val_top2_acc: 0.9979
Epoch 13/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0327 -
accuracy: 0.9892 - top2_acc: 0.9997 - val_loss: 0.0880 - val_accuracy: 0.9769 -
val_top2_acc: 0.9972
Epoch 14/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0307 -
accuracy: 0.9900 - top2_acc: 0.9997 - val_loss: 0.0763 - val_accuracy: 0.9783 -
val_top2_acc: 0.9986
Epoch 15/24
724/724 [==============================] - 38s 53ms/step - loss: 0.0286 -
accuracy: 0.9903 - top2_acc: 0.9999 - val_loss: 0.0761 - val_accuracy: 0.9776 -
val_top2_acc: 0.9982
Epoch 16/24
724/724 [==============================] - 39s 54ms/step - loss: 0.0320 -
accuracy: 0.9883 - top2_acc: 0.9997 - val_loss: 0.0775 - val_accuracy: 0.9776 -
val_top2_acc: 0.9989
Epoch 17/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0293 -
accuracy: 0.9902 - top2_acc: 0.9997 - val_loss: 0.0676 - val_accuracy: 0.9769 -
val_top2_acc: 0.9979
Epoch 18/24
724/724 [==============================] - 39s 53ms/step - loss: 0.0247 -
accuracy: 0.9914 - top2_acc: 0.9997 - val_loss: 0.0720 - val_accuracy: 0.9793 -
val_top2_acc: 0.9989
Epoch 19/24
724/724 [==============================] - 39s 54ms/step - loss: 0.0247 -
accuracy: 0.9905 - top2_acc: 0.9997 - val_loss: 0.0832 - val_accuracy: 0.9765 -
val_top2_acc: 0.9982
Epoch 20/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0249 -
accuracy: 0.9916 - top2_acc: 0.9998 - val_loss: 0.0714 - val_accuracy: 0.9797 -
val_top2_acc: 0.9989
Epoch 21/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0248 -
accuracy: 0.9917 - top2_acc: 0.9997 - val_loss: 0.0751 - val_accuracy: 0.9804 -
val_top2_acc: 0.9986
Epoch 22/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0244 -
accuracy: 0.9914 - top2_acc: 0.9998 - val_loss: 0.0671 - val_accuracy: 0.9804 -
val_top2_acc: 0.9986
Epoch 23/24
724/724 [==============================] - 37s 51ms/step - loss: 0.0250 -
accuracy: 0.9917 - top2_acc: 0.9997 - val_loss: 0.0805 - val_accuracy: 0.9786 -
val_top2_acc: 0.9979
Epoch 24/24
```

```
724/724 [==============================] - 37s 51ms/step - loss: 0.0209 -
accuracy: 0.9930 - top2_acc: 0.9999 - val_loss: 0.0738 - val_accuracy: 0.9779 -
val_top2_acc: 0.9982
```

Visualizações:

[32]:

```
f = plt.figure(figsize=(15,5))
ax = f.add_subplot(121)
ax.plot(best_history.history['accuracy'])
ax.plot(best_history.history['val_accuracy'])
ax.set_title('Model Accuracy')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.legend(['Train', 'Val'])

ax2 = f.add_subplot(122)
ax2.plot(best_history.history['loss'])
ax2.plot(best_history.history['val_loss'])
ax2.set_title('Model Loss')
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend(['Train', 'Val'], loc='upper left')

plt.show()
```



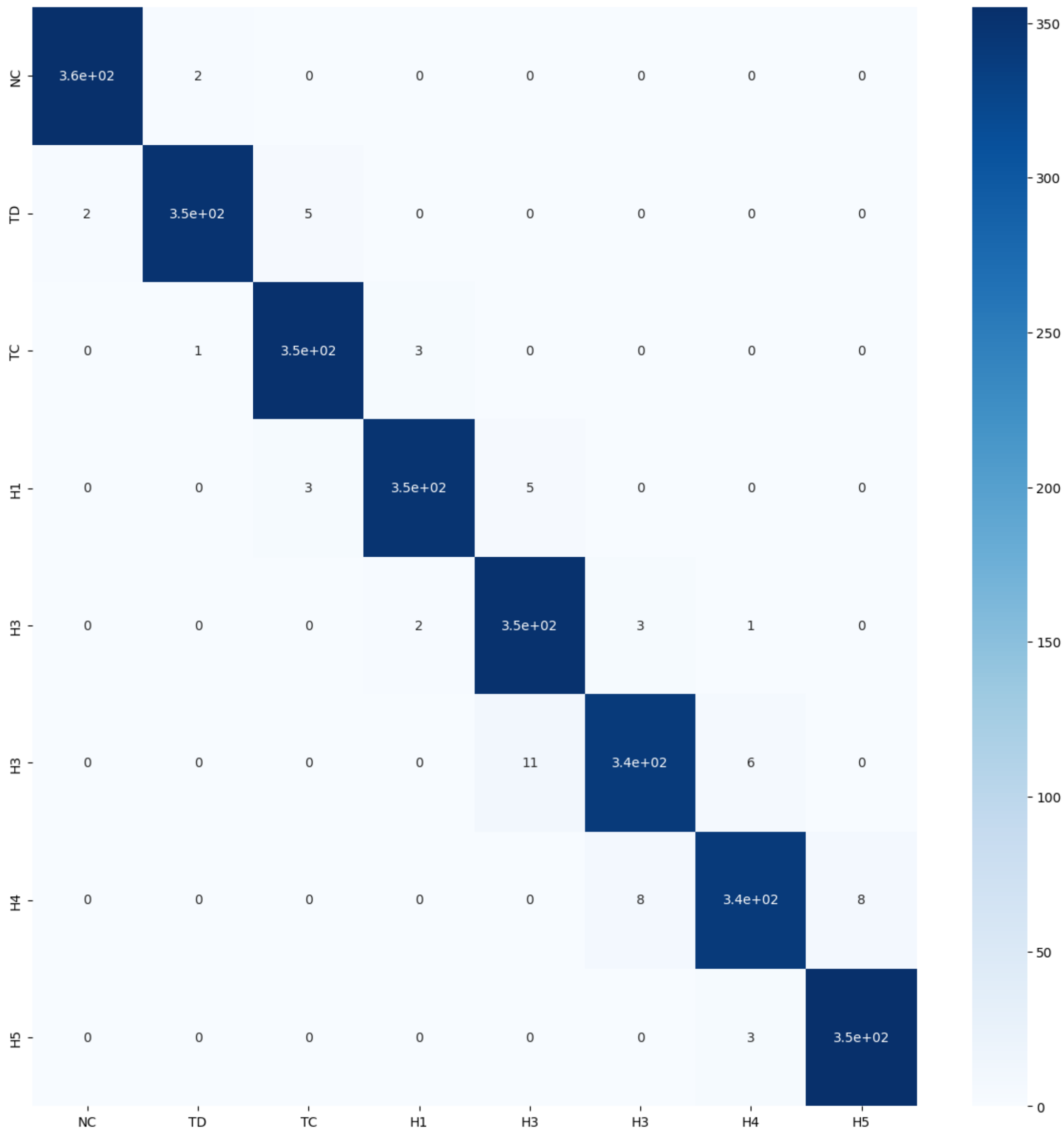
[33]:

```
pred = best_model.predict(val)
p = np.argmax(pred, axis=1)
y_valid = np.argmax(val_labels, axis=1, out=None)
results = confusion_matrix(y_valid, p)
classes=['NC','TD','TC','H1','H3','H3','H4','H5']
df_cm = pd.DataFrame(results, index = [i for i in classes], columns = [i for i␣
↪in classes])
plt.figure(figsize = (15,15))
```

```
sn.heatmap(df_cm, annot=True, cmap="Blues")
```

```
90/90 [==============================] - 3s 32ms/step
```

[33]: <Axes: >



Salvando o modelo:

[34]:
```
model.save("model_v3.h5")
```