Classificaço de Patologias usando Imagens Médicas

Carregar imagens do diretório

In [1]:

```
import os
current_dir = os.path.abspath(os.getcwd())
```

Converter base de dados para treino, validação e teste

In [2]:

```
#cria nova pasta para cachorros e gatos atendendo a estrutura do Keras/Tenso
folder = "/novo"
train_folder = current_dir + folder + "/train"
val_folder = current_dir + folder + "/val"
test_folder = current_dir + folder + "/test"

model_filepath = "keras/classificacao_02_03.keras"
conversao_path = "conversao/conversao_02_03"
```

Fazer o Tensorflow carregar as imagens para a RNA

In [3]:

```
import tensorflow as tf

print(tf.config.list_physical_devices('GPU'))
print(tf.__version__)
```

```
2022-06-29 07:46:47.347256: W tensorflow/stream_executor/platform/default/dso
_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: l
ibcudart.so.11.0: cannot open shared object file: No such file or directory
2022-06-29 07:46:47.347272: I tensorflow/stream_executor/cuda/cudart_stub.cc:
29] Ignore above cudart dlerror if you do not have a GPU set up on your machi
ne.
[]
2.6.1
```

```
2022-06-29 07:46:47.995773: W tensorflow/stream_executor/platform/default/dso
_loader.cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcud
a.so.1: cannot open shared object file: No such file or directory
2022-06-29 07:46:47.995789: W tensorflow/stream_executor/cuda/cuda_driver.cc:
269] failed call to cuInit: UNKNOWN ERROR (303)
2022-06-29 07:46:47.995800: I tensorflow/stream_executor/cuda/cuda_diagnostic
s.cc:156] kernel driver does not appear to be running on this host (pc): /pro
c/driver/nvidia/version does not exist
```

In [4]:

```
from tensorflow.keras.utils import image_dataset_from_directory
#image_dataset_from_directory monta uma estrutura de dados com imagens 180x1
# de 32 em 32 imagens
train_dataset = image_dataset_from_directory(train_folder, image_size=(180, 1

validation_dataset = image_dataset_from_directory(val_folder,image_size=(180,
test_dataset = image_dataset_from_directory(test_folder, image_size=(180, 18(
```

```
Found 34931 files belonging to 2 classes.
Found 16 files belonging to 2 classes.
Found 484 files belonging to 2 classes.
```

```
2022-06-29 07:46:48.493037: I tensorflow/core/platform/cpu_feature_guard.cc:1
42] This TensorFlow binary is optimized with oneAPI Deep Neural Network Libra
ry (oneDNN) to use the following CPU instructions in performance-critical ope
rations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate c
ompiler flags.
```

In [5]:

```
#
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    print(data_batch[0].shape)
    break
```

```
data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
(180, 180, 3)
```

```
2022-06-29 07:46:48.543356: I tensorflow/compiler/mlir/mlir_graph_optimizatio
n_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered
2)
```

Treinando o modelo

In [6]:

```
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.layers.experimental.preprocessing import Rescaling

#cria uma arquitetura de uma rede neural profunda vazia
model = keras.Sequential()
model.add(Rescaling(scale=1.0/255))
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(1, activation="sigmoid"))
model.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy'
#model.add(Dense(4, activation='softmax'))
#model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['ac
```

In [7]:

```
from tensorflow.keras.callbacks import ModelCheckpoint

callbacks = [
    ModelCheckpoint(
        filepath = model_filepath,
        save_best_only = True,
        monitor = "val_loss"
    )
]

history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
1092/1092 [==============================] - 350s 320ms/step - loss: 0.4272 -
accuracy: 0.8083 - val_loss: 0.1753 - val_accuracy: 0.9375
Epoch 2/50
1092/1092 [==============================] - 342s 313ms/step - loss: 0.2633 -
accuracy: 0.8977 - val_loss: 0.1099 - val_accuracy: 1.0000
Epoch 3/50
1092/1092 [==============================] - 341s 312ms/step - loss: 0.2064 -
accuracy: 0.9256 - val_loss: 0.0315 - val_accuracy: 1.0000
Epoch 4/50
1092/1092 [==============================] - 349s 319ms/step - loss: 0.1465 -
accuracy: 0.9501 - val_loss: 0.0277 - val_accuracy: 1.0000
Epoch 5/50
1092/1092 [==============================] - 343s 314ms/step - loss: 0.0887 -
accuracy: 0.9719 - val_loss: 0.0146 - val_accuracy: 1.0000
Epoch 6/50
1092/1092 [==============================] - 342s 313ms/step - loss: 0.0493 -
accuracy: 0.9853 - val_loss: 0.0093 - val_accuracy: 1.0000
Epoch 7/50
1092/1092 [==============================] - 342s 313ms/step - loss: 0.0257 -
accuracy: 0.9931 - val_loss: 0.0085 - val_accuracy: 1.0000
Epoch 8/50
1092/1092 [==============================] - 345s 316ms/step - loss: 0.0166 -
accuracy: 0.9962 - val_loss: 0.0079 - val_accuracy: 1.0000
Epoch 9/50
1092/1092 [==============================] - 345s 316ms/step - loss: 0.0133 -
accuracy: 0.9965 - val_loss: 0.0039 - val_accuracy: 1.0000
Epoch 10/50
1092/1092 [==============================] - 341s 312ms/step - loss: 0.0117 -
accuracy: 0.9968 - val_loss: 0.1475 - val_accuracy: 0.9375
Epoch 11/50
1092/1092 [==============================] - 346s 316ms/step - loss: 0.0094 -
accuracy: 0.9975 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 12/50
1092/1092 [==============================] - 356s 326ms/step - loss: 0.0086 -
accuracy: 0.9975 - val_loss: 0.1214 - val_accuracy: 0.9375
Epoch 13/50
1092/1092 [==============================] - 361s 330ms/step - loss: 0.0069 -
accuracy: 0.9984 - val_loss: 0.3659 - val_accuracy: 0.9375
Epoch 14/50
1092/1092 [==============================] - 346s 316ms/step - loss: 0.0083 -
accuracy: 0.9975 - val_loss: 0.4307 - val_accuracy: 0.9375
Epoch 15/50
1092/1092 [==============================] - 430s 394ms/step - loss: 0.0066 -
accuracy: 0.9979 - val_loss: 0.0425 - val_accuracy: 1.0000
Epoch 16/50
1092/1092 [==============================] - 408s 374ms/step - loss: 0.0072 -
accuracy: 0.9985 - val_loss: 6.5603e-04 - val_accuracy: 1.0000
Epoch 17/50
1092/1092 [==============================] - 392s 358ms/step - loss: 0.0065 -
accuracy: 0.9988 - val_loss: 6.8530e-04 - val_accuracy: 1.0000
Epoch 18/50
1092/1092 [==============================] - 416s 381ms/step - loss: 0.0073 -
accuracy: 0.9985 - val_loss: 0.0065 - val_accuracy: 1.0000
Epoch 19/50
1092/1092 [==============================] - 376s 344ms/step - loss: 0.0056 -
accuracy: 0.9987 - val_loss: 0.0044 - val_accuracy: 1.0000
Epoch 20/50
1092/1092 [==============================] - 343s 314ms/step - loss: 0.0038 -
accuracy: 0.9993 - val_loss: 0.0039 - val_accuracy: 1.0000
Epoch 21/50
```

```
1092/1092 [==============================] - 352s 322ms/step - loss: 0.0053 -
accuracy: 0.9989 - val_loss: 8.0096e-05 - val_accuracy: 1.0000
Epoch 22/50
1092/1092 [==============================] - 357s 326ms/step - loss: 0.0054 -
accuracy: 0.9987 - val_loss: 2.4788e-04 - val_accuracy: 1.0000
Epoch 23/50
1092/1092 [==============================] - 353s 323ms/step - loss: 0.0079 -
accuracy: 0.9981 - val_loss: 3.5050e-05 - val_accuracy: 1.0000
Epoch 24/50
1092/1092 [==============================] - 406s 372ms/step - loss: 0.0049 -
accuracy: 0.9991 - val_loss: 0.0109 - val_accuracy: 1.0000
Epoch 25/50
1092/1092 [==============================] - 328s 301ms/step - loss: 0.0035 -
accuracy: 0.9993 - val_loss: 2.7066e-05 - val_accuracy: 1.0000
Epoch 26/50
1092/1092 [==============================] - 384s 352ms/step - loss: 0.0058 -
accuracy: 0.9990 - val_loss: 7.4073e-06 - val_accuracy: 1.0000
Epoch 27/50
1092/1092 [==============================] - 418s 382ms/step - loss: 0.0039 -
accuracy: 0.9989 - val_loss: 0.0182 - val_accuracy: 1.0000
Epoch 28/50
1092/1092 [==============================] - 355s 324ms/step - loss: 0.0038 -
accuracy: 0.9993 - val_loss: 0.0106 - val_accuracy: 1.0000
Epoch 29/50
1092/1092 [==============================] - 361s 330ms/step - loss: 0.0031 -
accuracy: 0.9994 - val_loss: 0.0044 - val_accuracy: 1.0000
Epoch 30/50
1092/1092 [==============================] - 358s 328ms/step - loss: 0.0068 -
accuracy: 0.9986 - val_loss: 0.0713 - val_accuracy: 0.9375
Epoch 31/50
1092/1092 [==============================] - 335s 307ms/step - loss: 0.0039 -
accuracy: 0.9993 - val_loss: 3.5125e-06 - val_accuracy: 1.0000
Epoch 32/50
1092/1092 [==============================] - 343s 314ms/step - loss: 0.0048 -
accuracy: 0.9990 - val_loss: 3.6036e-04 - val_accuracy: 1.0000
Epoch 33/50
1092/1092 [==============================] - 348s 319ms/step - loss: 0.0033 -
accuracy: 0.9995 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 34/50
1092/1092 [==============================] - 350s 321ms/step - loss: 0.0051 -
accuracy: 0.9987 - val_loss: 8.2792e-07 - val_accuracy: 1.0000
Epoch 35/50
1092/1092 [==============================] - 334s 305ms/step - loss: 0.0029 -
accuracy: 0.9995 - val_loss: 2.8217e-05 - val_accuracy: 1.0000
Epoch 36/50
1092/1092 [==============================] - 338s 309ms/step - loss: 0.0022 -
accuracy: 0.9995 - val_loss: 7.7823e-05 - val_accuracy: 1.0000
Epoch 37/50
1092/1092 [==============================] - 342s 313ms/step - loss: 0.0062 -
accuracy: 0.9987 - val_loss: 0.2868 - val_accuracy: 0.9375
Epoch 38/50
1092/1092 [==============================] - 333s 305ms/step - loss: 0.0038 -
accuracy: 0.9993 - val_loss: 0.3457 - val_accuracy: 0.9375
Epoch 39/50
1092/1092 [==============================] - 334s 305ms/step - loss: 0.0028 -
accuracy: 0.9995 - val_loss: 0.0171 - val_accuracy: 1.0000
Epoch 40/50
1092/1092 [==============================] - 332s 304ms/step - loss: 0.0035 -
accuracy: 0.9993 - val_loss: 0.1829 - val_accuracy: 0.9375
Epoch 41/50
1092/1092 [==============================] - 333s 305ms/step - loss: 0.0033 -
accuracy: 0.9993 - val_loss: 0.0916 - val_accuracy: 0.9375
Epoch 42/50
1092/1092 [==============================] - 327s 299ms/step - loss: 0.0043 -
```

```
accuracy: 0.9993 - val_loss: 0.0384 - val_accuracy: 1.0000
Epoch 43/50
1092/1092 [==============================] - 326s 298ms/step - loss: 0.0037 -
accuracy: 0.9994 - val_loss: 0.0037 - val_accuracy: 1.0000
Epoch 44/50
1092/1092 [==============================] - 325s 297ms/step - loss: 0.0045 -
accuracy: 0.9991 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 45/50
1092/1092 [==============================] - 321s 294ms/step - loss: 0.0031 -
accuracy: 0.9995 - val_loss: 5.7260e-05 - val_accuracy: 1.0000
Epoch 46/50
1092/1092 [==============================] - 326s 298ms/step - loss: 0.0050 -
accuracy: 0.9993 - val_loss: 1.8811e-04 - val_accuracy: 1.0000
Epoch 47/50
1092/1092 [==============================] - 325s 297ms/step - loss: 0.0032 -
accuracy: 0.9995 - val_loss: 2.4629e-06 - val_accuracy: 1.0000
Epoch 48/50
1092/1092 [==============================] - 326s 298ms/step - loss: 0.0033 -
accuracy: 0.9994 - val_loss: 4.0221e-05 - val_accuracy: 1.0000
Epoch 49/50
1092/1092 [==============================] - 325s 298ms/step - loss: 0.0034 -
accuracy: 0.9995 - val_loss: 1.0507e-04 - val_accuracy: 1.0000
Epoch 50/50
1092/1092 [==============================] - 332s 304ms/step - loss: 0.0026 -
accuracy: 0.9996 - val_loss: 2.6431e-04 - val_accuracy: 1.0000
```

In [8]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
rescaling (Rescaling)       (None, 180, 180, 3)       0

_________________________________________________________________
conv2d (Conv2D)             (None, 178, 178, 32)      896

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 89, 89, 32)       0

_________________________________________________________________
conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

_________________________________________________________________
flatten (Flatten)           (None, 484416)            0

_________________________________________________________________
dense (Dense)               (None, 1)                 484417

=================================================================
Total params: 503,809
Trainable params: 503,809
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

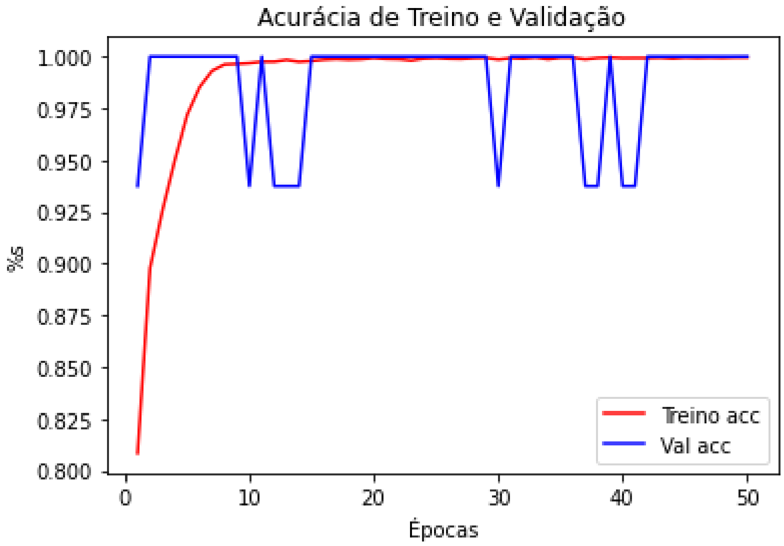
```
#https://www.tensorflow.org/js/tutorials/conversion/import_keras?hl=pt-br#al
import tensorflowjs as tfjs
tfjs.converters.save_keras_model(model, conversao_path)
```

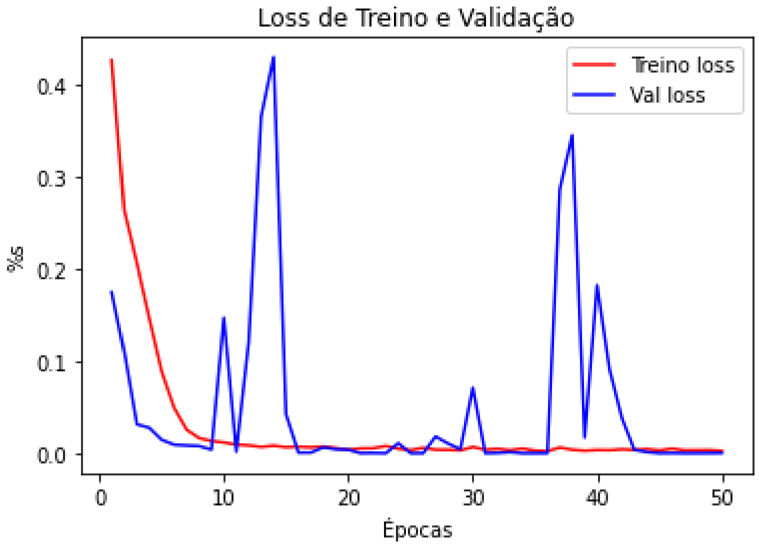
Visualização de Resultados

In [10]:

```
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
```

```
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "r", label="Treino acc")
plt.plot(epochs, val_accuracy, "b", label="Val acc")
plt.xlabel("Épocas")
plt.ylabel("%s")
plt.title("Acurácia de Treino e Validação")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "r", label="Treino loss")
plt.plot(epochs, val_loss, "b", label="Val loss")
plt.xlabel("Épocas")
plt.ylabel("%s")
plt.title("Loss de Treino e Validação")
plt.legend()
plt.show()
```





Resultados do Conjunto de Teste

In [11]:

```
from tensorflow import keras
model = keras.models.load_model(model_filepath)
```

In [12]:

```
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
16/16 [==============================] - 4s 71ms/step - loss: 0.0034 - accura
cy: 1.0000
Test accuracy: 1.000
```

Referências

- https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
- https://stackoverflow.com/questions/3430372/how-do-i-get-the-full-path-of-the-current-files-directory
- https://www.geeksforgeeks.org/python-list-files-in-a-directory/
- https://pynative.com/python-random-sample/
- https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
- https://www.mygreatlearning.com/blog/keras-tutorial/
- https://www.machinecurve.com/index.php/2020/03/30/how-to-use-conv2d-with-keras/
- https://www.pyimagesearch.com/2021/06/30/how-to-use-the-modelcheckpoint-callback-with-keras-and-tensorflow/