Classificaço de Patologias usando Imagens Médicas

Carregar imagens do diretório

In [13]:

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

Converter base de dados para treino, validação e teste

In [14]:

```
#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_04.keras"
conversao_path = "conversao/conversao_02_04"
```

Fazer o Tensorflow carregar as imagens para a RNA

In [15]:

```
import tensorflow as tf

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

```
[]
2.6.1
```

In [16]:

```
from tensorflow.keras.utils import image_dataset_from_directory
#image_dataset_from_directory monta uma estrutura de dados com imagens 180x18
# 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.
```

In [17]:

```
#
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)
```

Treinando o modelo

In [18]:

```
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 [19]:

```
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 [==============================] - 332s 302ms/step - loss: 0.4469 -
accuracy: 0.7973 - val_loss: 0.2108 - val_accuracy: 0.9375
Epoch 2/50
1092/1092 [==============================] - 323s 296ms/step - loss: 0.2793 -
accuracy: 0.8911 - val_loss: 0.2251 - val_accuracy: 0.9375
Epoch 3/50
1092/1092 [==============================] - 322s 295ms/step - loss: 0.2248 -
accuracy: 0.9166 - val_loss: 0.1096 - val_accuracy: 1.0000
Epoch 4/50
1092/1092 [==============================] - 321s 294ms/step - loss: 0.1744 -
accuracy: 0.9372 - val_loss: 0.1754 - val_accuracy: 0.9375
Epoch 5/50
1092/1092 [==============================] - 322s 294ms/step - loss: 0.1183 -
accuracy: 0.9594 - val_loss: 0.1688 - val_accuracy: 0.9375
Epoch 6/50
1092/1092 [==============================] - 321s 294ms/step - loss: 0.0699 -
accuracy: 0.9776 - val_loss: 0.0136 - val_accuracy: 1.0000
Epoch 7/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0413 -
accuracy: 0.9875 - val_loss: 0.0168 - val_accuracy: 1.0000
Epoch 8/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0239 -
```

```
accuracy: 0.9930 - val_loss: 0.0068 - val_accuracy: 1.0000
Epoch 9/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0175 -
accuracy: 0.9949 - val_loss: 0.0097 - val_accuracy: 1.0000
Epoch 10/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0152 -
accuracy: 0.9952 - val_loss: 0.0077 - val_accuracy: 1.0000
Epoch 11/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0125 -
accuracy: 0.9967 - val_loss: 0.0364 - val_accuracy: 1.0000
Epoch 12/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0094 -
accuracy: 0.9979 - val_loss: 0.0035 - val_accuracy: 1.0000
Epoch 13/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0117 -
accuracy: 0.9967 - val_loss: 0.0057 - val_accuracy: 1.0000
Epoch 14/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0099 -
accuracy: 0.9974 - val_loss: 0.0064 - val_accuracy: 1.0000
Epoch 15/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0098 -
accuracy: 0.9971 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 16/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0069 -
accuracy: 0.9981 - val_loss: 0.0091 - val_accuracy: 1.0000
Epoch 17/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0097 -
accuracy: 0.9974 - val_loss: 0.0159 - val_accuracy: 1.0000
Epoch 18/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0103 -
accuracy: 0.9973 - val_loss: 0.0296 - val_accuracy: 1.0000
Epoch 19/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0056 -
accuracy: 0.9987 - val_loss: 0.1110 - val_accuracy: 0.9375
Epoch 20/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0076 -
accuracy: 0.9982 - val_loss: 0.0100 - val_accuracy: 1.0000
Epoch 21/50
1092/1092 [==============================] - 318s 292ms/step - loss: 0.0066 -
accuracy: 0.9982 - val_loss: 0.0945 - val_accuracy: 0.9375
Epoch 22/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0057 -
accuracy: 0.9984 - val_loss: 0.0535 - val_accuracy: 1.0000
Epoch 23/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0096 -
accuracy: 0.9977 - val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 24/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0089 -
accuracy: 0.9984 - val_loss: 0.0100 - val_accuracy: 1.0000
Epoch 25/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0063 -
accuracy: 0.9990 - val_loss: 1.0457e-04 - val_accuracy: 1.0000
Epoch 26/50
1092/1092 [==============================] - 318s 291ms/step - loss: 0.0071 -
accuracy: 0.9985 - val_loss: 0.0037 - val_accuracy: 1.0000
Epoch 27/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0065 -
accuracy: 0.9991 - val_loss: 1.3024e-04 - val_accuracy: 1.0000
Epoch 28/50
1092/1092 [==============================] - 318s 291ms/step - loss: 0.0062 -
accuracy: 0.9991 - val_loss: 0.0218 - val_accuracy: 1.0000
Epoch 29/50
1092/1092 [==============================] - 317s 290ms/step - loss: 0.0052 -
accuracy: 0.9992 - val_loss: 7.3197e-05 - val_accuracy: 1.0000
```

```
Epoch 30/50
1092/1092 [==============================] - 317s 290ms/step - loss: 0.0045 -
accuracy: 0.9991 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 31/50
1092/1092 [==============================] - 317s 291ms/step - loss: 0.0053 -
accuracy: 0.9989 - val_loss: 6.7052e-05 - val_accuracy: 1.0000
Epoch 32/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0059 -
accuracy: 0.9989 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 33/50
1092/1092 [==============================] - 318s 291ms/step - loss: 0.0061 -
accuracy: 0.9987 - val_loss: 0.0194 - val_accuracy: 1.0000
Epoch 34/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0046 -
accuracy: 0.9992 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 35/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0051 -
accuracy: 0.9993 - val_loss: 8.0291e-05 - val_accuracy: 1.0000
Epoch 36/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0051 -
accuracy: 0.9990 - val_loss: 7.9099e-05 - val_accuracy: 1.0000
Epoch 37/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0043 -
accuracy: 0.9994 - val_loss: 5.2471e-05 - val_accuracy: 1.0000
Epoch 38/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0060 -
accuracy: 0.9988 - val_loss: 1.2637e-04 - val_accuracy: 1.0000
Epoch 39/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0034 -
accuracy: 0.9993 - val_loss: 2.5867e-04 - val_accuracy: 1.0000
Epoch 40/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0064 -
accuracy: 0.9989 - val_loss: 0.0203 - val_accuracy: 1.0000
Epoch 41/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0050 -
accuracy: 0.9991 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 42/50
1092/1092 [==============================] - 320s 293ms/step - loss: 0.0062 -
accuracy: 0.9989 - val_loss: 0.0079 - val_accuracy: 1.0000
Epoch 43/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0039 -
accuracy: 0.9993 - val_loss: 0.0102 - val_accuracy: 1.0000
Epoch 44/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0041 -
accuracy: 0.9993 - val_loss: 4.1379e-04 - val_accuracy: 1.0000
Epoch 45/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0045 -
accuracy: 0.9991 - val_loss: 8.8540e-04 - val_accuracy: 1.0000
Epoch 46/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0040 -
accuracy: 0.9993 - val_loss: 1.7399e-04 - val_accuracy: 1.0000
Epoch 47/50
1092/1092 [==============================] - 318s 291ms/step - loss: 0.0035 -
accuracy: 0.9993 - val_loss: 9.8218e-05 - val_accuracy: 1.0000
Epoch 48/50
1092/1092 [==============================] - 318s 291ms/step - loss: 0.0043 -
accuracy: 0.9996 - val_loss: 2.1355e-04 - val_accuracy: 1.0000
Epoch 49/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0047 -
accuracy: 0.9989 - val_loss: 4.0590e-04 - val_accuracy: 1.0000
Epoch 50/50
1092/1092 [==============================] - 319s 292ms/step - loss: 0.0049 -
accuracy: 0.9991 - val_loss: 1.2089e-04 - val_accuracy: 1.0000
```

In [20]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
flatten_1 (Flatten)          (None, 484416)            0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 484417
=================================================================
Total params: 503,809
Trainable params: 503,809
Non-trainable params: 0
_________________________________________________________________
```

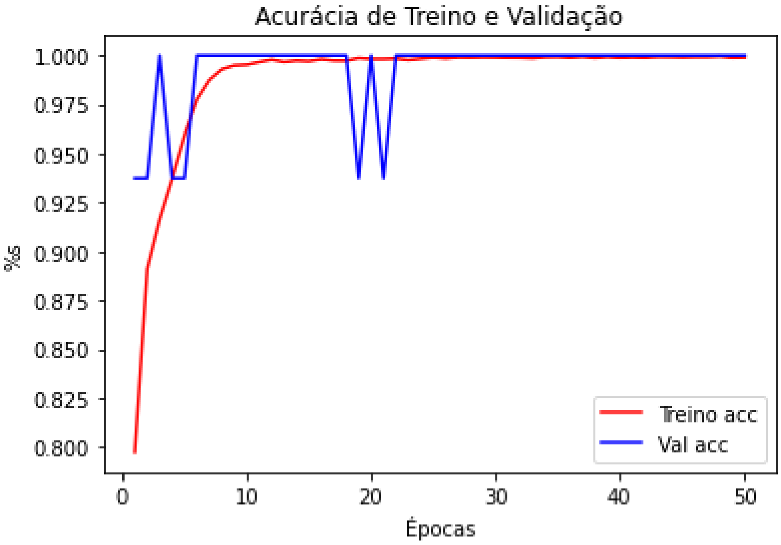
In [21]:

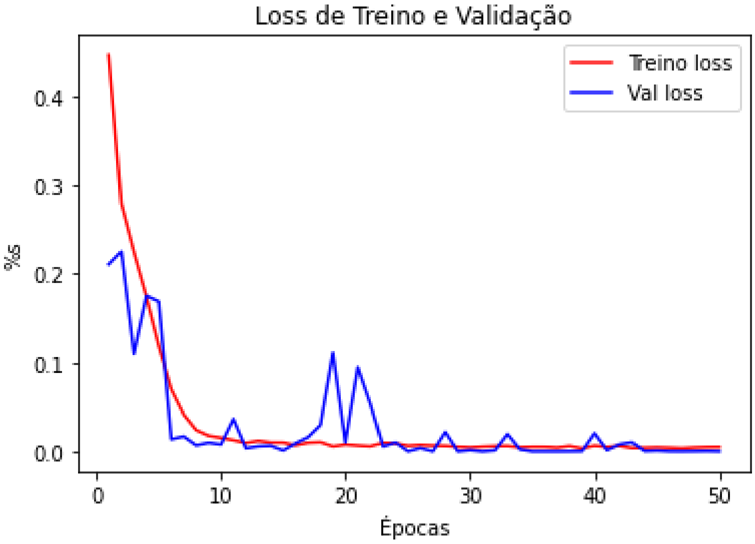
```
#https://www.tensorflow.org/js/tutorials/conversion/import_keras?hl=pt-br#al
import tensorflowjs as tfjs
tfjs.converters.save_keras_model(model, conversao_path)
```

Visualização de Resultados

In [22]:

```
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 [23]:
```
from tensorflow import keras
model = keras.models.load_model(model_filepath)
```

In [24]:
```
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
16/16 [==============================] - 2s 66ms/step - loss: 0.0200 - accura
cy: 0.9959
Test accuracy: 0.996
```

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/