# Classificaço de Patologias usando Imagens Médicas

## Carregar imagens do diretório

In [37]:
```
import os
current_dir = os.path.abspath(os.getcwd())
```

## Converter base de dados para treino, validação e teste

In [38]:
```
#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_06.keras"
conversao_path = "conversao/conversao_02_06"
```

# Fazer o Tensorflow carregar as imagens para a RNA

In [39]:
```
import tensorflow as tf

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

```
[]
2.6.1
```

In [40]:
```
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, 180
```

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

In [41]:
```
#
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 [42]:

```
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 [43]:

```
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 [==============================] - 370s 338ms/step - loss: 0.4402 -
accuracy: 0.8030 - val_loss: 0.2502 - val_accuracy: 0.9375
Epoch 2/50
1092/1092 [==============================] - 354s 324ms/step - loss: 0.2940 -
accuracy: 0.8835 - val_loss: 0.1336 - val_accuracy: 1.0000
Epoch 3/50
1092/1092 [==============================] - 369s 338ms/step - loss: 0.2487 -
accuracy: 0.9063 - val_loss: 0.0828 - val_accuracy: 1.0000
Epoch 4/50
1092/1092 [==============================] - 349s 320ms/step - loss: 0.2083 -
accuracy: 0.9243 - val_loss: 0.0422 - val_accuracy: 1.0000
Epoch 5/50
1092/1092 [==============================] - 352s 322ms/step - loss: 0.1607 -
accuracy: 0.9441 - val_loss: 0.2115 - val_accuracy: 0.9375
Epoch 6/50
1092/1092 [==============================] - 362s 331ms/step - loss: 0.1108 -
accuracy: 0.9624 - val_loss: 0.1343 - val_accuracy: 0.9375
Epoch 7/50
1092/1092 [==============================] - 356s 326ms/step - loss: 0.0682 -
accuracy: 0.9784 - val_loss: 0.1117 - val_accuracy: 0.9375
Epoch 8/50
1092/1092 [==============================] - 328s 300ms/step - loss: 0.0390 -
```

```
accuracy: 0.9879 - val_loss: 0.0109 - val_accuracy: 1.0000
Epoch 9/50
1092/1092 [==============================] - 325s 298ms/step - loss: 0.0237 -
accuracy: 0.9932 - val_loss: 0.0050 - val_accuracy: 1.0000
Epoch 10/50
1092/1092 [==============================] - 326s 298ms/step - loss: 0.0167 -
accuracy: 0.9952 - val_loss: 1.7613e-04 - val_accuracy: 1.0000
Epoch 11/50
1092/1092 [==============================] - 343s 314ms/step - loss: 0.0119 -
accuracy: 0.9969 - val_loss: 0.1054 - val_accuracy: 0.9375
Epoch 12/50
1092/1092 [==============================] - 352s 322ms/step - loss: 0.0094 -
accuracy: 0.9975 - val_loss: 3.1622e-04 - val_accuracy: 1.0000
Epoch 13/50
1092/1092 [==============================] - 346s 317ms/step - loss: 0.0132 -
accuracy: 0.9962 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 14/50
1092/1092 [==============================] - 349s 319ms/step - loss: 0.0113 -
accuracy: 0.9966 - val_loss: 0.0386 - val_accuracy: 1.0000
Epoch 15/50
1092/1092 [==============================] - 354s 324ms/step - loss: 0.0099 -
accuracy: 0.9972 - val_loss: 0.2162 - val_accuracy: 0.8750
Epoch 16/50
1092/1092 [==============================] - 356s 326ms/step - loss: 0.0076 -
accuracy: 0.9981 - val_loss: 3.9273e-04 - val_accuracy: 1.0000
Epoch 17/50
1092/1092 [==============================] - 356s 325ms/step - loss: 0.0079 -
accuracy: 0.9975 - val_loss: 0.0736 - val_accuracy: 0.9375
Epoch 18/50
1092/1092 [==============================] - 353s 322ms/step - loss: 0.0098 -
accuracy: 0.9971 - val_loss: 0.0027 - val_accuracy: 1.0000
Epoch 19/50
1092/1092 [==============================] - 346s 315ms/step - loss: 0.0071 -
accuracy: 0.9981 - val_loss: 0.0107 - val_accuracy: 1.0000
Epoch 20/50
1092/1092 [==============================] - 347s 318ms/step - loss: 0.0076 -
accuracy: 0.9979 - val_loss: 2.2447e-04 - val_accuracy: 1.0000
Epoch 21/50
1092/1092 [==============================] - 348s 319ms/step - loss: 0.0071 -
accuracy: 0.9984 - val_loss: 3.4981e-04 - val_accuracy: 1.0000
Epoch 22/50
1092/1092 [==============================] - 350s 320ms/step - loss: 0.0065 -
accuracy: 0.9987 - val_loss: 0.0069 - val_accuracy: 1.0000
Epoch 23/50
1092/1092 [==============================] - 361s 330ms/step - loss: 0.0071 -
accuracy: 0.9981 - val_loss: 5.3955e-05 - val_accuracy: 1.0000
Epoch 24/50
1092/1092 [==============================] - 358s 328ms/step - loss: 0.0065 -
accuracy: 0.9989 - val_loss: 7.2903e-04 - val_accuracy: 1.0000
Epoch 25/50
1092/1092 [==============================] - 356s 324ms/step - loss: 0.0077 -
accuracy: 0.9983 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 26/50
1092/1092 [==============================] - 347s 318ms/step - loss: 0.0051 -
accuracy: 0.9990 - val_loss: 4.3147e-04 - val_accuracy: 1.0000
Epoch 27/50
1092/1092 [==============================] - 348s 318ms/step - loss: 0.0048 -
accuracy: 0.9991 - val_loss: 2.6674e-04 - val_accuracy: 1.0000
Epoch 28/50
1092/1092 [==============================] - 350s 321ms/step - loss: 0.0060 -
accuracy: 0.9991 - val_loss: 6.9639e-04 - val_accuracy: 1.0000
Epoch 29/50
1092/1092 [==============================] - 341s 311ms/step - loss: 0.0048 -
accuracy: 0.9988 - val_loss: 0.1343 - val_accuracy: 0.9375
```

```
Epoch 30/50
1092/1092 [==============================] - 341s 311ms/step - loss: 0.0039 -
accuracy: 0.9993 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 31/50
1092/1092 [==============================] - 341s 312ms/step - loss: 0.0062 -
accuracy: 0.9983 - val_loss: 0.0115 - val_accuracy: 1.0000
Epoch 32/50
1092/1092 [==============================] - 346s 316ms/step - loss: 0.0050 -
accuracy: 0.9989 - val_loss: 0.0193 - val_accuracy: 1.0000
Epoch 33/50
1092/1092 [==============================] - 346s 316ms/step - loss: 0.0039 -
accuracy: 0.9993 - val_loss: 0.0469 - val_accuracy: 0.9375
Epoch 34/50
1092/1092 [==============================] - 329s 300ms/step - loss: 0.0050 -
accuracy: 0.9989 - val_loss: 7.2636e-04 - val_accuracy: 1.0000
Epoch 35/50
1092/1092 [==============================] - 338s 308ms/step - loss: 0.0046 -
accuracy: 0.9990 - val_loss: 1.4807e-04 - val_accuracy: 1.0000
Epoch 36/50
1092/1092 [==============================] - 340s 311ms/step - loss: 0.0053 -
accuracy: 0.9992 - val_loss: 0.0164 - val_accuracy: 1.0000
Epoch 37/50
1092/1092 [==============================] - 332s 303ms/step - loss: 0.0049 -
accuracy: 0.9991 - val_loss: 0.1760 - val_accuracy: 0.9375
Epoch 38/50
1092/1092 [==============================] - 338s 309ms/step - loss: 0.0058 -
accuracy: 0.9989 - val_loss: 9.1092e-04 - val_accuracy: 1.0000
Epoch 39/50
1092/1092 [==============================] - 366s 336ms/step - loss: 0.0048 -
accuracy: 0.9993 - val_loss: 1.5977e-05 - val_accuracy: 1.0000
Epoch 40/50
1092/1092 [==============================] - 367s 336ms/step - loss: 0.0054 -
accuracy: 0.9990 - val_loss: 0.0085 - val_accuracy: 1.0000
Epoch 41/50
1092/1092 [==============================] - 345s 316ms/step - loss: 0.0044 -
accuracy: 0.9990 - val_loss: 5.6202e-06 - val_accuracy: 1.0000
Epoch 42/50
1092/1092 [==============================] - 348s 318ms/step - loss: 0.0056 -
accuracy: 0.9989 - val_loss: 5.3802e-05 - val_accuracy: 1.0000
Epoch 43/50
1092/1092 [==============================] - 353s 323ms/step - loss: 0.0050 -
accuracy: 0.9990 - val_loss: 1.1488e-04 - val_accuracy: 1.0000
Epoch 44/50
1092/1092 [==============================] - 328s 300ms/step - loss: 0.0048 -
accuracy: 0.9991 - val_loss: 6.1955e-05 - val_accuracy: 1.0000
Epoch 45/50
1092/1092 [==============================] - 337s 309ms/step - loss: 0.0027 -
accuracy: 0.9996 - val_loss: 9.7290e-05 - val_accuracy: 1.0000
Epoch 46/50
1092/1092 [==============================] - 331s 303ms/step - loss: 0.0037 -
accuracy: 0.9993 - val_loss: 0.0442 - val_accuracy: 0.9375
Epoch 47/50
1092/1092 [==============================] - 339s 310ms/step - loss: 0.0024 -
accuracy: 0.9997 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 48/50
1092/1092 [==============================] - 339s 310ms/step - loss: 0.0044 -
accuracy: 0.9993 - val_loss: 5.0210e-04 - val_accuracy: 1.0000
Epoch 49/50
1092/1092 [==============================] - 339s 310ms/step - loss: 0.0029 -
accuracy: 0.9996 - val_loss: 2.9255e-04 - val_accuracy: 1.0000
Epoch 50/50
1092/1092 [==============================] - 339s 310ms/step - loss: 0.0047 -
accuracy: 0.9993 - val_loss: 5.1547e-06 - val_accuracy: 1.0000
```

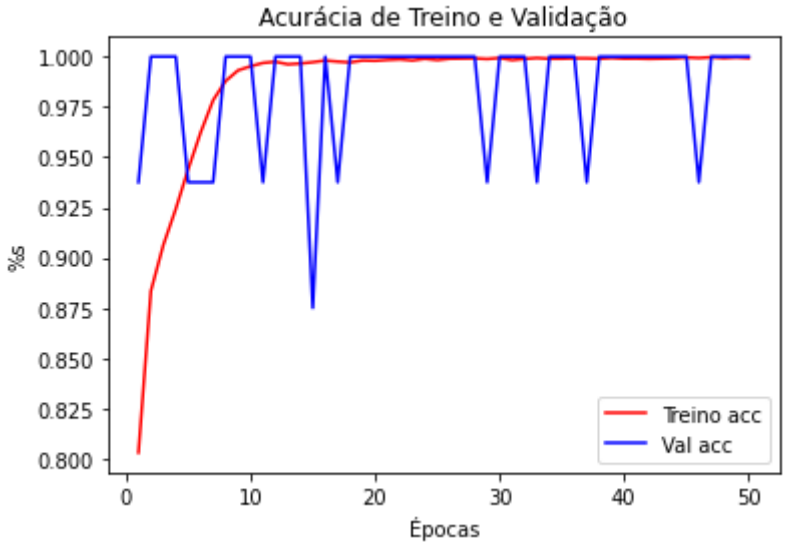
In [44]:
```
model.summary()
```

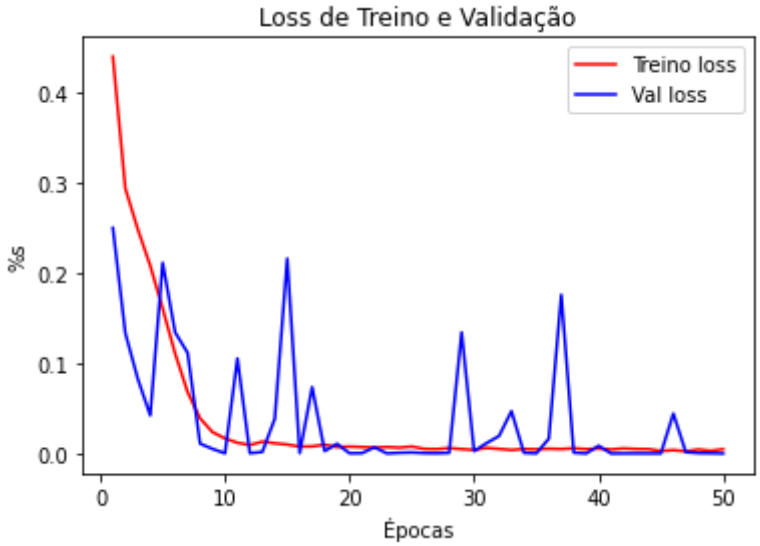
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_3 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
flatten_3 (Flatten)          (None, 484416)            0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 484417
=================================================================
Total params: 503,809
Trainable params: 503,809
Non-trainable params: 0
_________________________________________________________________
```

In [45]:
```
#https://www.tensorflow.org/js/tutorials/conversion/import_keras?hl=pt-br#ali
import tensorflowjs as tfjs
tfjs.converters.save_keras_model(model, conversao_path)
```

# Visualização de Resultados

In [46]:
```
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 [47]:
```
from tensorflow import keras
model = keras.models.load_model(model_filepath)
```

In [48]:
```
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
16/16 [==============================] - 3s 63ms/step - loss: 0.0212 - accura
cy: 0.9938
Test accuracy: 0.994
```

# 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/