Redes Neurais Convolucionais

Nesta competição, você escreverá um algoritmo para classificar se as imagens contêm um cachorro ou um gato. Isso é fácil para humanos, cães e gatos. Seu computador terá um pouco mais de dificuldade.

Problema: Dada uma imagem, é um cachorro ou um gato?

Para essa tarefa, usaremos uma arquitetura de Rede Neural Convolucional. Essa arquitetura usa métodos de convolução para poder prever características específicas de uma imagem de acordo com o que aprende em um conjunto de treinamento. Por exemplo, podemos dizer que é possível perceber a diferença ao procurar bigodes em um gato ou focinho comprido em um cachorro. Mas uma Rede Neural Convolucional procura muitos outros recursos baseados no que temos em um conjunto de treinamento.

Conjunto de dados de treino: 2.000 imagens de cães e 2.000 imagens de gatos.

Conjunto de dados de validação: 500 imagens de cães e 500 imagens de gatos.

Conjunto de dados de teste: Teremos x imagens de cães e gatos.

Essa á uma questão onde os iniciantes tem muitas dúvidas. Por que precisamos de dados de treino, validação e teste? Usamos os dados de treino para treinar o algoritmo e então criar o modelo preditivo. Usamos os dados de validação, para avaliar o modelo durante o treinamento. Usamos os dados de teste para validar a performance do modelo já treinado, ou seja, apresentamos ao modelo dados que ele não viu durante o treinamento, a fim de garantir que ele é capaz de fazer previsões.

Vamos começar!

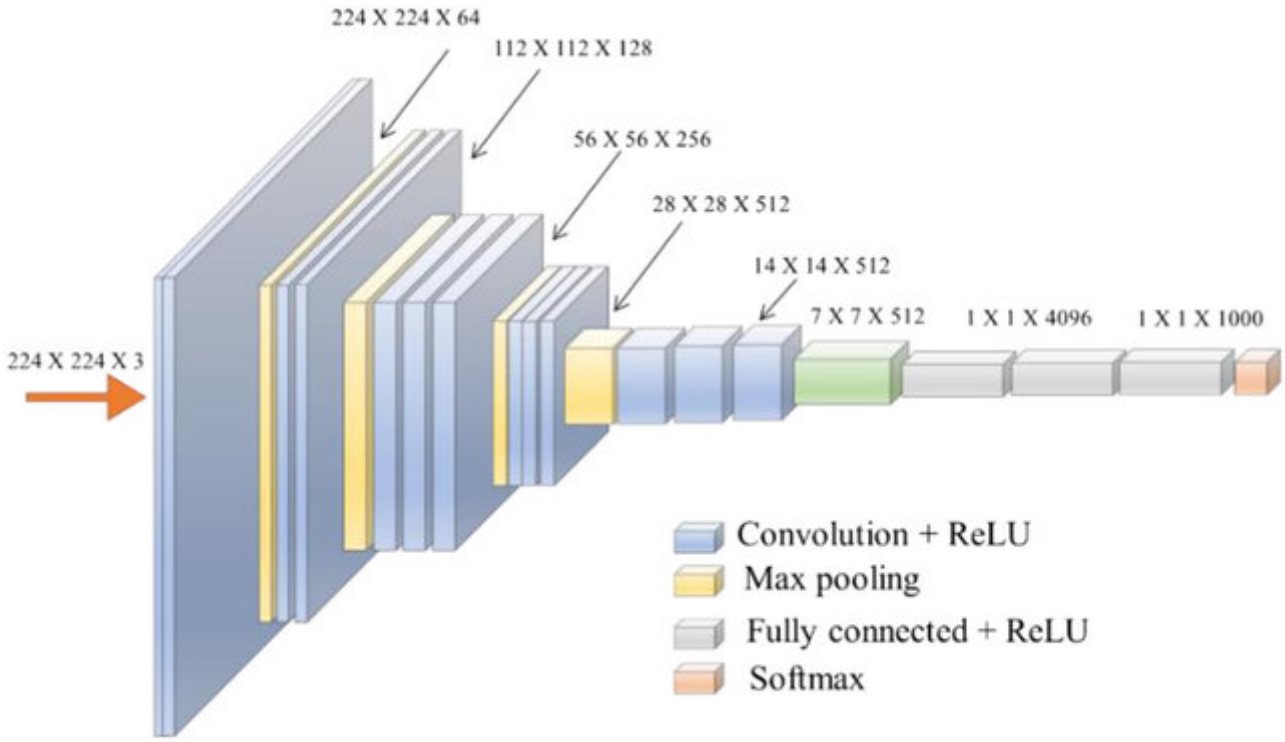
Importando as bibliotecas

In [1]:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from keras.preprocessing import image
```

```
Using TensorFlow backend.
```

Arquitetura da Rede Convolucional



In [2]:

```
classificador = Sequential()
classificador.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_unif
                 padding='same', input_shape=(224, 224, 3)))
classificador.add(BatchNormalization())
classificador.add(MaxPooling2D((2, 2)))
classificador.add(Dropout(0.2))

classificador.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_unif
classificador.add(BatchNormalization())
classificador.add(MaxPooling2D((2, 2)))
classificador.add(Dropout(0.25))

classificador.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uni
classificador.add(BatchNormalization())
classificador.add(MaxPooling2D((2, 2)))
classificador.add(Dropout(0.3))

classificador.add(Flatten())
classificador.add(Dense(1024, activation='relu', kernel_initializer='he_uniform'))
classificador.add(BatchNormalization())
classificador.add(Dropout(0.35))
classificador.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
classificador.add(BatchNormalization())
classificador.add(Dropout(0.4))
classificador.add(Dense(2, activation='relu', kernel_initializer='he_uniform'))
classificador.add(BatchNormalization())
classificador.add(Dropout(0.5))
classificador.add(Dense(1, activation='sigmoid'))

classificador.compile(optimizer = 'adam', loss = 'binary_crossentropy',
                      metrics = ['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\aluno\Anaconda3\lib\site-packages\ker
as\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is depr
ecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From C:\Users\aluno\Anaconda3\lib\site-packages\ten
sorflow\python\ops\nn_impl.py:180: add_dispatch_support.<locals>.wrapp
er (from tensorflow.python.ops.array_ops) is deprecated and will be re
moved in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

Vamos gerar algumas imagens

Normalizar as imagens

In [3]:

```
gerador_treinamento = ImageDataGenerator(rescale = 1./255,
                                         rotation_range = 7,
                                         horizontal_flip = True,
                                         shear_range = 0.2,
                                         height_shift_range = 0.07,
                                         zoom_range = 0.2)
gerador_teste = ImageDataGenerator(rescale = 1./255)
```

In [4]:

```
base_treinamento = gerador_treinamento.flow_from_directory('dataset/training_set',
                                                           target_size = (224, 224),
                                                           batch_size = 64,
                                                           class_mode = 'binary')
```

```
Found 4000 images belonging to 2 classes.
```

In [5]:

```
base_teste = gerador_teste.flow_from_directory('dataset/test_set',
                                               target_size = (224, 224),
                                               batch_size = 64,
                                               class_mode = 'binary')
```

```
Found 1000 images belonging to 2 classes.
```

In [6]:

```
classificador.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 32)      896
_________________________________________________________________
batch_normalization_1 (Batch (None, 224, 224, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 32)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 112, 112, 64)      256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 128)       73856
_________________________________________________________________
batch_normalization_3 (Batch (None, 56, 56, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 28, 28, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 100352)            0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              102761472
_________________________________________________________________
batch_normalization_4 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               131200
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
_________________________________________________________________
batch_normalization_6 (Batch (None, 2)                 8
_________________________________________________________________
dropout_6 (Dropout)          (None, 2)                 0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 3
=================================================================
Total params: 102,991,693
Trainable params: 102,988,937
Non-trainable params: 2,756
_________________________________________________________________
```

Treinamento

In [7]:

```
classificador.fit_generator(base_treinamento, steps_per_epoch = 40,
                            epochs = 10, validation_data = base_teste,
                            validation_steps = 10)
```

```
WARNING:tensorflow:From C:\Users\aluno\Anaconda3\lib\site-packages\ker
as\backend\tensorflow_backend.py:422: The name tf.global_variables is
deprecated. Please use tf.compat.v1.global_variables instead.

Epoch 1/10
40/40 [==============================] - 327s 8s/step - loss: 0.7390 -
accuracy: 0.5554 - val_loss: 1.8502 - val_accuracy: 0.5078
Epoch 2/10
40/40 [==============================] - 295s 7s/step - loss: 0.7081 -
accuracy: 0.5520 - val_loss: 2.1666 - val_accuracy: 0.5114
Epoch 3/10
40/40 [==============================] - 298s 7s/step - loss: 0.6669 -
accuracy: 0.5934 - val_loss: 2.4428 - val_accuracy: 0.4969
Epoch 4/10
40/40 [==============================] - 301s 8s/step - loss: 0.6462 -
accuracy: 0.6040 - val_loss: 1.1604 - val_accuracy: 0.4968
Epoch 5/10
40/40 [==============================] - 310s 8s/step - loss: 0.6290 -
accuracy: 0.6187 - val_loss: 0.9963 - val_accuracy: 0.5081
Epoch 6/10
40/40 [==============================] - 322s 8s/step - loss: 0.6294 -
accuracy: 0.6171 - val_loss: 1.1165 - val_accuracy: 0.5531
Epoch 7/10
40/40 [==============================] - 340s 8s/step - loss: 0.6083 -
accuracy: 0.6325 - val_loss: 0.7945 - val_accuracy: 0.5032
Epoch 8/10
40/40 [==============================] - 330s 8s/step - loss: 0.6143 -
accuracy: 0.6418 - val_loss: 0.9489 - val_accuracy: 0.5341
Epoch 9/10
40/40 [==============================] - 233s 6s/step - loss: 0.6038 -
accuracy: 0.6590 - val_loss: 0.6681 - val_accuracy: 0.5531
Epoch 10/10
40/40 [==============================] - 220s 5s/step - loss: 0.6090 -
accuracy: 0.6566 - val_loss: 0.5904 - val_accuracy: 0.6607
```

Out[7]:

```
<keras.callbacks.callbacks.History at 0x28cfe427eb8>
```

Teste *

In [56]:

```
imagem_teste = image.load_img('dataset/test_set/cachorro/dog.3969.jpg',
                              target_size = (224,224))
```

In [57]:

```
imagem_teste
```

Out[57]:

In [58]:

```
imagem_teste = image.img_to_array(imagem_teste)
imagem_teste /= 255
```

In [59]:

```
imagem_teste = np.expand_dims(imagem_teste, axis = 0)
```

In [60]:

```
previsao = classificador.predict(imagem_teste)
previsao
```

Out[60]:

```
array([[0.26230136]], dtype=float32)
```

In [61]:

```
base_treinamento.class_indices
```

Out[61]:

```
{'cachorro': 0, 'gato': 1}
```

In [62]:

```
previsao = (previsao > 0.5)
previsao
```

Out[62]:

```
array([[False]])
```

In [63]:

```
previsao
```

Out[63]:

```
array([[False]])
```

In [ ]:

```

```