T4_Gerador_dados

December 3, 2020

Trabalho #4 - Gerador de dados

Nesse trabalho você vai treinar uma RNA para realizar uma tarefa de classificação de múltiplas classes. A tarefa consiste em identificar três tipos de animais: gato, cachorro e panda. Esse problema foi proposto no Kaggle em 2019 e pode ser acessado em https://www.kaggle.com/ashishsaxena2209/animal-image-datasetdog-cat-and-panda.

A diferença principal desse trabalho em relação aos outros já realizados até o momento, é a utilização de imagens reias, que possuem dimensões e proporções diferentes, objetos não centrados, luminosidades diferentes etc.

Nesse trabalho para processar as imagens de forma a normalizá-las e redimensioná-las para que tenham dimensão uniforme são usados geradores de dados. Além disso, para eliminar problemas de "overfitting" é também parte desse trabalho treinar uma RNA com geração artificial de dados.

Esse trabalho é dividio nas seguintes etapas:

1. Explorar as imagens do conjunto de dados;
2. Construir e treinar uma RNA para identificar o animal mostrado na imagem;
3. Treinar uma nova RNA para identificar o animal mostrado usando geração artificial de dados;
4. Avaliar e comparar o desempenho das duas RNAs.

1 Coloque o seu nome aqui:

Nome: Bruno Rodrigues Silva

2 Imagens do conjunto de dados

A primeira etapa do trabalho é carregar o conjunto de dados, que consiste em um arquipo tipo zip de 3.000 fotos no formato JPG de gatos, cães e pandas, e extrair localmente no diretório `tmp`.

NOTA: As 3.000 imagens usadas neste trabalho foram extraídas do conjunto de dados "Dogs-Cats-Pandas", disponível no Kaggle, no link https://www.kaggle.com/ashishsaxena2209/animal-image-datasetdog-cat-and-panda.

2.1 Carregar arquivo de dados para o Colab

Execute a célula abaixo para carregar as imagens para o seu Colab. Após a execução dessa célula as imagens estão no arquivo `cats_dogs_pandas.zip` no diteório `tmp` do seu ambiente do Colab.

Importante: se você estiver usando o notebook Jupiter do Anaconda essa célula não deve ser executada e o arquivo compactado com os dados deve estar no subdiretório `tmp` do diretório onde se encontra o seu notebook.

[1]:
```
!pip install gdown
!gdown --id 1BSbYML5rw6srpxNEd9mdVhzfkS605Sfd -O /tmp/cat_dog_panda.zip
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.6/dist-packages
(3.6.4)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-
packages (from gdown) (2.23.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
(from gdown) (4.41.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from gdown) (1.15.0)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.6/dist-packages (from requests->gdown) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests->gdown) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests->gdown) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.6/dist-packages (from requests->gdown) (2020.11.8)
Downloading...
From: https://drive.google.com/uc?id=1BSbYML5rw6srpxNEd9mdVhzfkS605Sfd
To: /tmp/cat_dog_panda.zip
197MB [00:01, 103MB/s]
```

Execute a célula abaixo para descompactar o arquivo com as imagens. O código usa a biblioteca `os`, que possui funções do sistema operacional que fornecem acesso ao sistema de arquivos, e a biblioteca zipfile que permite descompactar arquivos.

[2]:
```
import os
import zipfile

local_zip = '/tmp/cat_dog_panda.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')

zip_ref.extractall('/tmp')
zip_ref.close()
```

O conteúdo do arquivo `cat_dog_panda.zip` é extraído para o diretório `content/tmp/cat_dog_panda`, que contém os subdiretórios `train`, `val` e `test` com as imagens dos conjuntos de dados de treinamento, validação e teste.

Lembre que, conforme visto na aula, o gerador `ImageDataGenerator` do Keras identifica e cataloga as imagens automaticamente a partir dos subdiretórios onde elas se encontram. Assim, por exemplo, no diretório `train` existe um diretório `cats`, um ditetório `dogs` e outro `pandas`. O

`ImageGenerator` rotula automaticamente as imagens, simplificando a etapa de codificação dos dados.

Importante:

O Keras fornece rótulos para classes de acordo com a ordem que os subdiretórios estão nos diretórios `train`, `val` e `pandas`. Assim, se o subdiretório `cats` é o primeiro então, os gatos serão a classe 0.

2.2 Exercíco #1: Definir os nomes dos diretórios

Complete a célula de código abaixo para criar variáveis com os nomes dos diretórios e subdiretórios com as imagens de treinamento, validação e teste. Para isso utilize a função `path.join(diretório_base, subdiretório)` da biblioteca `os` (https://docs.python.org/2/library/os.path.html).

[3]:
```
# PARA VOCÊ FAZER: definir nomes dos diretórios

import os

# Nome do diretório base
base_dir = '/tmp/cat_dog_panda'

# Path dos diretórios de treinamento, validação e teste
# Incluir seu código aqui
#
train_cats_dir = os.path.join(base_dir, 'train/cats')
val_cats_dir = os.path.join(base_dir, 'val/cats')
test_cats_dir = os.path.join(base_dir, 'test/cats')
# Path dos subdiretórios com as imagens do dados de treinamento
# Incluir seu código aqui
#
train_dogs_dir = os.path.join(base_dir, 'train/dogs')
val_dogs_dir = os.path.join(base_dir, 'val/dogs')
test_dogs_dir = os.path.join(base_dir, 'test/dogs')
# Path dos subdiretórios com as imagens do dados de validação
# Incluir seu código aqui
#

train_pandas_dir = os.path.join(base_dir, 'train/pandas')
val_pandas_dir = os.path.join(base_dir, 'val/pandas')
test_pandas_dir = os.path.join(base_dir, 'test/pandas')
# Path dos subdiretórios com as imagens do dados de teste
# Incluir seu código aqui
#

print('Subdiretório de imagens de gatos para treinamento =', train_cats_dir)
print('Subdiretório de imagens de cães para treinamento =', train_dogs_dir)
print('Subdiretório de imagens de pandas para treinamento =', train_pandas_dir)
```

```
print('Subdiretório de imagens de gatos para validação =', val_cats_dir)
print('Subdiretório de imagens de cães para validação =', val_dogs_dir)
print('Subdiretório de imagens de pandas para validação =', val_pandas_dir)
print('Subdiretório de imagens de gatos para teste =', test_cats_dir)
print('Subdiretório de imagens de cães para teste =', test_dogs_dir)
print('Subdiretório de imagens de pandas para teste =', test_pandas_dir)
```

```
Subdiretório de imagens de gatos para treinamento =
/tmp/cat_dog_panda/train/cats
Subdiretório de imagens de cães para treinamento = /tmp/cat_dog_panda/train/dogs
Subdiretório de imagens de pandas para treinamento =
/tmp/cat_dog_panda/train/pandas
Subdiretório de imagens de gatos para validação = /tmp/cat_dog_panda/val/cats
Subdiretório de imagens de cães para validação = /tmp/cat_dog_panda/val/dogs
Subdiretório de imagens de pandas para validação = /tmp/cat_dog_panda/val/pandas
Subdiretório de imagens de gatos para teste = /tmp/cat_dog_panda/test/cats
Subdiretório de imagens de cães para teste = /tmp/cat_dog_panda/test/dogs
Subdiretório de imagens de pandas para teste = /tmp/cat_dog_panda/test/pandas
```

Saída esperada:

```
Subdiretório de imagens de gatos para treinamento = /tmp/cat_dog_panda/train/cats
Subdiretório de imagens de cães para treinamento = /tmp/cat_dog_panda/train/dogs
Subdiretório de imagens de pandas para treinamento = /tmp/cat_dog_panda/train/pandas
Subdiretório de imagens de gatos para validação = /tmp/cat_dog_panda/val/cats
Subdiretório de imagens de cães para validação = /tmp/cat_dog_panda/val/dogs
Subdiretório de imagens de pandas para validação = /tmp/cat_dog_panda/val/pandas
Subdiretório de imagens de gatos para teste = /tmp/cat_dog_panda/test/cats
Subdiretório de imagens de cães para teste = /tmp/cat_dog_panda/test/dogs
Subdiretório de imagens de pandas para teste = /tmp/cat_dog_panda/test/pandas
```

2.3 Exercíco #2: Criar listas dos arquivos de imagens de treinamento e verificar número de exemplos

Modifique a célula de código abaixo para criar listas com os nomes dos arquivos nos subdiretórios `cats` edogs `train` do diretório de treinamento. Esses nomes serão utilizados para acessar as imagens de forma a permitir a sua visualização e análise. Para realizar essa tarefa utilize a função `listdir(diretório)` da biblioteca `os` (https://docs.python.org/2/library/os.html?highlight=listdir#os.listdir).

Após criar essa lista, verifique o número total de imagens em cada subdiretório dos diretórios de treinamento, validação e teste. Para isso use a função `len` do python para calcular o número de elementos de uma lista.

[8]:
```
# PARA VOCÊ FAZER: Criar lista dos arquivos dos subdiretórios do diretório de␣
↪treinamento e verificar número de exemplos

# Listas de arquivos imagens de treinamento
# Incluir seu código aqui
```

```
train_cat_fnames = os.listdir(train_cats_dir)
train_dog_fnames = os.listdir(train_dogs_dir)
train_panda_fnames = os.listdir(train_pandas_dir)

# Calcular número de exemplos de treinamento
# Incluir seu código aqui
#
len_cat_train = len(os.listdir(train_cats_dir))
len_dog_train = len(os.listdir(train_dogs_dir))
len_panda_train = len(os.listdir(train_pandas_dir))
# Calcular número de exemplos de validação, usar len(os.lisdir(diretório))
# Incluir seu código aqui
#

len_cat_val = len(os.listdir(val_cats_dir))
len_dog_val = len(os.listdir(val_dogs_dir))
len_panda_val = len(os.listdir(val_pandas_dir))
# Calcular número de exemplos de teste, usar len(os.lisdir(diretório))
# Incluir seu código aqui
#

len_cat_test = len(os.listdir(test_cats_dir))
len_dog_test = len(os.listdir(test_dogs_dir))
len_panda_test = len(os.listdir(test_pandas_dir))
print('Nomes dos arquivos de gatos (5 primeiros):', train_cat_fnames[:5])
print('Nomes dos arquivos de cães (5 primeiros):', train_dog_fnames[:5])
print('Nomes dos arquivos de pandas (5 primeiros):', train_panda_fnames[:5])

print('Total imagens treinamento gatos:', len_cat_train)
print('Total imagens treinamento cães:', len_dog_train)
print('Total imagens treinamento pandas:', len_panda_train)

print('Total imagens validação gatos :', len_cat_val)
print('Total imagens validação cães:', len_dog_val)
print('Total imagens validação pandas:', len_panda_val)

print('Total imagens teste gatos:', len_cat_test)
print('Total imagens teste cães:', len_dog_test)
print('Total imagens teste pandas:', len_panda_test)
```

```
Nomes dos arquivos de gatos (5 primeiros): ['cats_00741.jpg', 'cats_00953.jpg',
'cats_00929.jpg', 'cats_00770.jpg', 'cats_00964.jpg']
Nomes dos arquivos de cães (5 primeiros): ['dogs_00491.jpg', 'dogs_00794.jpg',
'dogs_00406.jpg', 'dogs_00682.jpg', 'dogs_00987.jpg']
Nomes dos arquivos de pandas (5 primeiros): ['panda_00699.jpg',
'panda_00846.jpg', 'panda_00725.jpg', 'panda_00744.jpg', 'panda_00856.jpg']
Total imagens treinamento gatos: 660
```

```
Total imagens treinamento cães: 660
Total imagens treinamento pandas: 660
Total imagens validação gatos : 170
Total imagens validação cães: 170
Total imagens validação pandas: 170
Total imagens teste gatos: 170
Total imagens teste cães: 170
Total imagens teste pandas: 170
```

Saída esperada:

```
Nomes dos arquivos de gatos (5 primeiros): ['cats_00800.jpg', 'cats_00998.jpg', 'cats_00403.jpg'
Nomes dos arquivos de cães (5 primeiros): ['dogs_00365.jpg', 'dogs_00523.jpg', 'dogs_00763.jpg',
Nomes dos arquivos de pandas (5 primeiros): ['panda_00637.jpg', 'panda_00774.jpg', 'panda_00874.
Total imagens treinamento gatos: 660
Total imagens treinamento cães: 660
Total imagens treinamento pandas: 660
Total imagens validação gatos : 170
Total imagens validação cães: 170
Total imagens validação pandas: 170
Total imagens teste gatos: 170
Total imagens teste cães: 170
Total imagens teste pandas: 170
```

Observe que existem 660 imagens de treinamento, 170 imagens de validação e 170 imagens de teste para cada animal. Ou seja, existem um total de 1980 imagens de treinamento, 510 imagens de validação e 510 imagens de teste.

2.4 Visalização das imagens

Execute a célula abaixo para visualizar algumas imagens de gatos, cães e pandas de treinamento.

[9]:
```
%matplotlib inline
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Número de linhas e colunas do arranjo para mostrar as imagens
nrows = 6
ncols = 4

# Define figura do matplotlib e define o tamanho para ser mostrada
fig = plt.gcf()
fig.set_size_inches(16, 16)

pic_index = 8

next_cat_pix = [os.path.join(train_cats_dir, fname)
                for fname in train_cat_fnames[pic_index-8:pic_index]]
next_dog_pix = [os.path.join(train_dogs_dir, fname)
```

```
                for fname in train_dog_fnames[pic_index-8:pic_index]]
next_panda_pix = [os.path.join(train_pandas_dir, fname)
                for fname in train_panda_fnames[ pic_index-8:pic_index]]

for i, img_path in enumerate(next_cat_pix+next_dog_pix+next_panda_pix):
    # Define índice da imagem
    sp = plt.subplot(nrows, ncols, i + 1)
    sp.axis('Off') # Não mostra eixos ou grids

    img = mpimg.imread(img_path)
    plt.imshow(img)

plt.show()
```

Observe que as imagens possuem formas e proporções diferente, dessa forma, antes de treinar uma rede Neural com essas imagens tem que ajustar as suas dimensões.

3 Pré-processamento dos dados

Nesse trabalho, vamos utilizar três geradores de dados para carregar as imagens dos diretórios de origem e convertê-las em tensores `float32`. Teremos um gerador para as imagens de treinamento, um para as imagens de validação e outro para as imagens de teste. Os geradores devem produzir lotes de 30 imagens, com dimensão 150x150, para classificação multiclasse.

Como você já sabe, os dados de entrada para a RNA devem ser normalizados, no caso de imagens o mais comum é ter as imagens normalizadas de forma a transformar os valores dos pixels, originalmente um número inteiro no intervalo [0, 255], para um número real no intervalo [0, 1].

3.1 Exercício #3: Pré-processamento de dados

Para criar e configurar os três geradores você vai usar a classe `ImageDataGenerator` do Keras com o parâmetro `rescale`. A classe `ImageDataGenerator` permite instanciar geradores de lotes de imagens, juntamente com os seus rótulos, usando o método `flow_from_directory (diretório)`. Esses geradores podem então ser usados com os métodos do Keras para treinamento, avaliação e previsão: `fit`,`evaluate` e `predict`.

Os três geradores devem ser configurados e instanciados da seguinte forma:

- Normalização dos pixles para valores no intervalo [0, 1]
- Tamanho do lote = 30
- Tipo de problema: classificação multiclasse
- Dimensão das imagens: 150x150

[12]:
```
# PARA VOCÊ FAZER: criar e instanciar os geradores de dados de treinamento, validação e teste

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define dimensão das imagens
# Incluir seu código aqui
#
img_dim = (150, 150)
# Cria gerador usando a classe ImageDataGenerator que normaliza imagens
# Incluir seu código aqui
datagen = ImageDataGenerator(rescale=1/255)

# Instancia gerador de imagens de treinamentocom o método flow_from (utilize a variável que define o diretório de dados de treinamento)
# Incluir seu código aqui
train_generator = datagen.flow_from_directory(os.path.join(base_dir, 'train'), target_size=img_dim,class_mode='categorical',batch_size=30)
```

```
# Instancia gerador de imagens de validação (utilize a variável que define o␣
 ↪diretório de dados de validação)
# Incluir seu código aqui
val_generator = datagen.flow_from_directory(os.path.join(base_dir, 'val'),␣
 ↪target_size=img_dim,class_mode='categorical',batch_size=30)



# Instancia gerador de imagens de teste (utilize a variável que define o␣
 ↪diretório de dados de teste)
# Incluir seu código aqui
test_generator = datagen.flow_from_directory(os.path.join(base_dir, 'test'),␣
 ↪target_size=img_dim,class_mode='categorical',batch_size=30)
```

```
Found 1980 images belonging to 3 classes.
Found 510 images belonging to 3 classes.
Found 510 images belonging to 3 classes.
```

Saída esperada:

```
Found 1980 images belonging to 3 classes.
Found 510 images belonging to 3 classes.
Found 510 images belonging to 3 classes.
```

4 Construir e treinar uma RNA para classificação

Para identificar o animal mostrado na imagem você utilizar uma RNA convolucional relativamente simples. Na medida em que as imagens estão em arquivos e possuem dimensões diferentes é necessário usar um gerador de dados para o treinamento da rede.

Nessa etapa do trabalho você vai configurar uma RNA convolucional, criar geradores de dados para carregar e processar as imagens, e finalmente treinar a RNA.

4.1 Exercício #4: Criação da RNA

Para resolver esse problema de classificação multiclasse, você vai usar uma RNA com 3 camadas convolucionais, seguidas de camadas "max-pooling", e 2 camadas densas, com as seguintes características:

- Dimensão das imagens: 150x150x3;
- Primeira camada convolucional: número de filtros 32, dimensão do filtro 3, função de ativação ReLu;
- Segunda camada convolucional: número de filtros 64, dimensão do filtro 3, função de ativação ReLu;
- Terceira camada convolucional: número de filtros 128, dimensão do filtro 3, função de ativação ReLu;
- Camadas de max-pooling: dimensão da janela 2, "stride" 2;
- Primeira camada densa: número de neurônios 256, função de ativação ReLu;

- Camada de saída: número de neurônio 3, função de ativação softmax.

Ressalta-se que após cada camada convolucional tem-se uma camada de max-pooling.

Na célula abaixo crie uma funçao que configura uma RNA com as características acima. Não se esqueça de incluir a camada de "flattening" entre a última camada de max-pooling e a primeira camada densa.

[15]:
```
# PARA VOCÊ FAZER: Função para criar RNA convolucional para classificação␣
 ↪multiclasse

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

def build_model(img_size):

    # Criação e configuração da RNA
    # Incluir seu código aqui
    #
    model = Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=img_size),
    layers.MaxPool2D((2,2), strides=2),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPool2D((2,2), strides=2),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPool2D((2,2), strides=2),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(3, activation='softmax')

    ])
    return model
```

[17]:
```
# PARA VOCÊ FAZER: Criar RNA convolucional para classificação dos animais

# Define dimensão das imagens
# Incluir seu código aqui
#
img_size=(150, 150, 3)
# Criação da RNA usando a função buil_model
# Incluir seu código aqui
#
rna = build_model(img_size)
rna.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
conv2d_3 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 36992)             0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               9470208
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 771
=================================================================
Total params: 9,564,227
Trainable params: 9,564,227
Non-trainable params: 0
_________________________________________________________________
```

Saída esperada:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 36992)             0
_________________________________________________________________
dense (Dense)                (None, 256)               9470208
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 771
=================================================================
Total params: 9,564,227
```

```
Trainable params: 9,564,227
Non-trainable params: 0
_________________________________________________________________
```

4.2 Exercício #5: Compilação e treinamento da RNA

Agora você vai treinar a sua RNA usando o método de otimização Adams. Assim, na célula abaixo, compile e treine a sua RNA usando os seguinte hiperparâmetros:

- método de otimização: Adam;
- taxa de aprendizagem = 0.001;
- número de épocas = 20;
- verbose = 2.

Cuidado para definir os parâmetros `steps_per_epoch` e `validation_steps` de forma a utilizar todos as iamgens de treinamento e de validação. Lembre que temos 1980 imagens de treinamento, 510 imagens de validação e o tamanho dos lotes é de 30 imagens.

Para treinamento da RNA, utilize o método `fit_generator` e os geradores de dados de treinamento e validação.

Não se esqueça de definir a função de custo apropriada para classificação multiclasse e escolher a métrica exatidão.

[19]:
```
# PARA VOCÊ FAZER: compilar e treinar a RNA

from tensorflow.keras import optimizers

# Comiplação da RNA
# Incluir seu código aqui
opt = optimizers.Adam(0.001)
rna.compile(opt, loss='categorical_crossentropy', metrics='accuracy')
# Treinamento da RNA
# Incluir seu código aqui
history = rna.fit(train_generator, epochs=20, steps_per_epoch=66,␣
 ↪validation_steps=17, validation_data=val_generator)
```

```
Epoch 1/20
66/66 [==============================] - 9s 137ms/step - loss: 0.9183 -
accuracy: 0.5424 - val_loss: 0.8674 - val_accuracy: 0.5725
Epoch 2/20
66/66 [==============================] - 9s 136ms/step - loss: 0.6729 -
accuracy: 0.6611 - val_loss: 0.6862 - val_accuracy: 0.6490
Epoch 3/20
66/66 [==============================] - 9s 136ms/step - loss: 0.6165 -
accuracy: 0.6955 - val_loss: 0.6678 - val_accuracy: 0.7020
Epoch 4/20
66/66 [==============================] - 9s 135ms/step - loss: 0.4776 -
accuracy: 0.7884 - val_loss: 0.6521 - val_accuracy: 0.6980
Epoch 5/20
```

```
66/66 [==============================] - 9s 135ms/step - loss: 0.3915 -
accuracy: 0.8207 - val_loss: 0.6360 - val_accuracy: 0.7098
Epoch 6/20
66/66 [==============================] - 9s 136ms/step - loss: 0.2578 -
accuracy: 0.8894 - val_loss: 0.8382 - val_accuracy: 0.6941
Epoch 7/20
66/66 [==============================] - 9s 135ms/step - loss: 0.1751 -
accuracy: 0.9354 - val_loss: 1.0005 - val_accuracy: 0.7137
Epoch 8/20
66/66 [==============================] - 9s 136ms/step - loss: 0.1364 -
accuracy: 0.9515 - val_loss: 0.9146 - val_accuracy: 0.7020
Epoch 9/20
66/66 [==============================] - 9s 135ms/step - loss: 0.1123 -
accuracy: 0.9571 - val_loss: 1.1025 - val_accuracy: 0.6863
Epoch 10/20
66/66 [==============================] - 9s 134ms/step - loss: 0.0546 -
accuracy: 0.9813 - val_loss: 1.2400 - val_accuracy: 0.6902
Epoch 11/20
66/66 [==============================] - 9s 135ms/step - loss: 0.0446 -
accuracy: 0.9889 - val_loss: 1.8410 - val_accuracy: 0.6765
Epoch 12/20
66/66 [==============================] - 9s 134ms/step - loss: 0.1459 -
accuracy: 0.9576 - val_loss: 1.2318 - val_accuracy: 0.6922
Epoch 13/20
66/66 [==============================] - 9s 134ms/step - loss: 0.0255 -
accuracy: 0.9949 - val_loss: 1.4553 - val_accuracy: 0.7000
Epoch 14/20
66/66 [==============================] - 9s 138ms/step - loss: 0.0112 -
accuracy: 0.9980 - val_loss: 1.5935 - val_accuracy: 0.7078
Epoch 15/20
66/66 [==============================] - 9s 135ms/step - loss: 0.0045 -
accuracy: 0.9995 - val_loss: 1.7525 - val_accuracy: 0.7000
Epoch 16/20
66/66 [==============================] - 9s 133ms/step - loss: 0.0013 -
accuracy: 1.0000 - val_loss: 1.8453 - val_accuracy: 0.7059
Epoch 17/20
66/66 [==============================] - 9s 134ms/step - loss: 6.6886e-04 -
accuracy: 1.0000 - val_loss: 1.8994 - val_accuracy: 0.7098
Epoch 18/20
66/66 [==============================] - 9s 134ms/step - loss: 4.1006e-04 -
accuracy: 1.0000 - val_loss: 1.9629 - val_accuracy: 0.7059
Epoch 19/20
66/66 [==============================] - 9s 134ms/step - loss: 3.2022e-04 -
accuracy: 1.0000 - val_loss: 2.0081 - val_accuracy: 0.7039
Epoch 20/20
66/66 [==============================] - 9s 135ms/step - loss: 2.5642e-04 -
accuracy: 1.0000 - val_loss: 2.0621 - val_accuracy: 0.7059
```

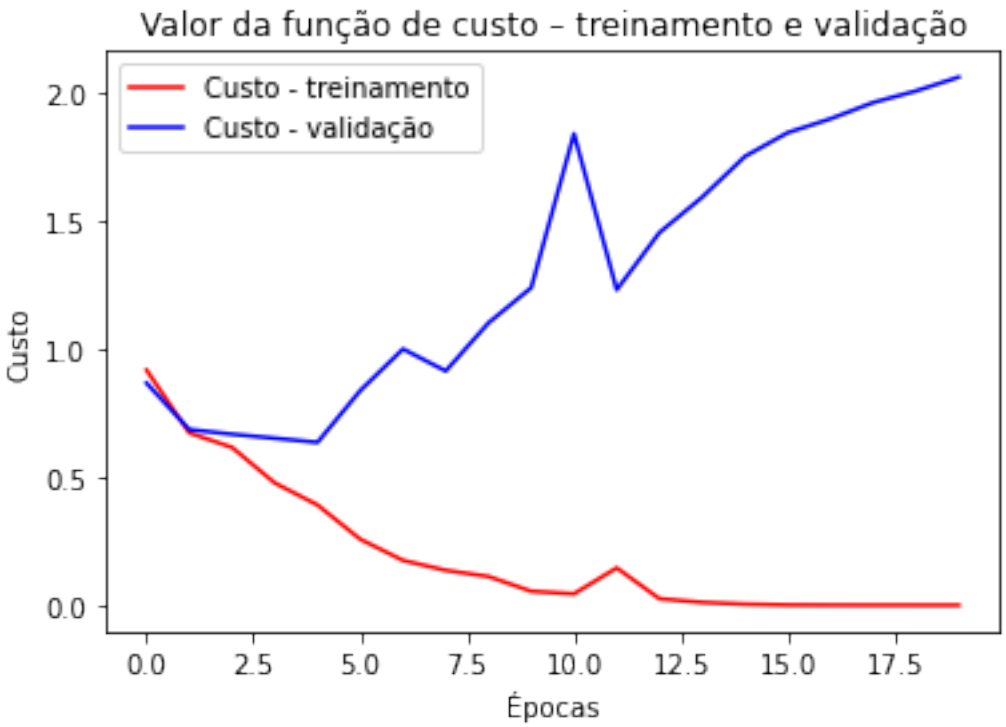
Saída esperada:

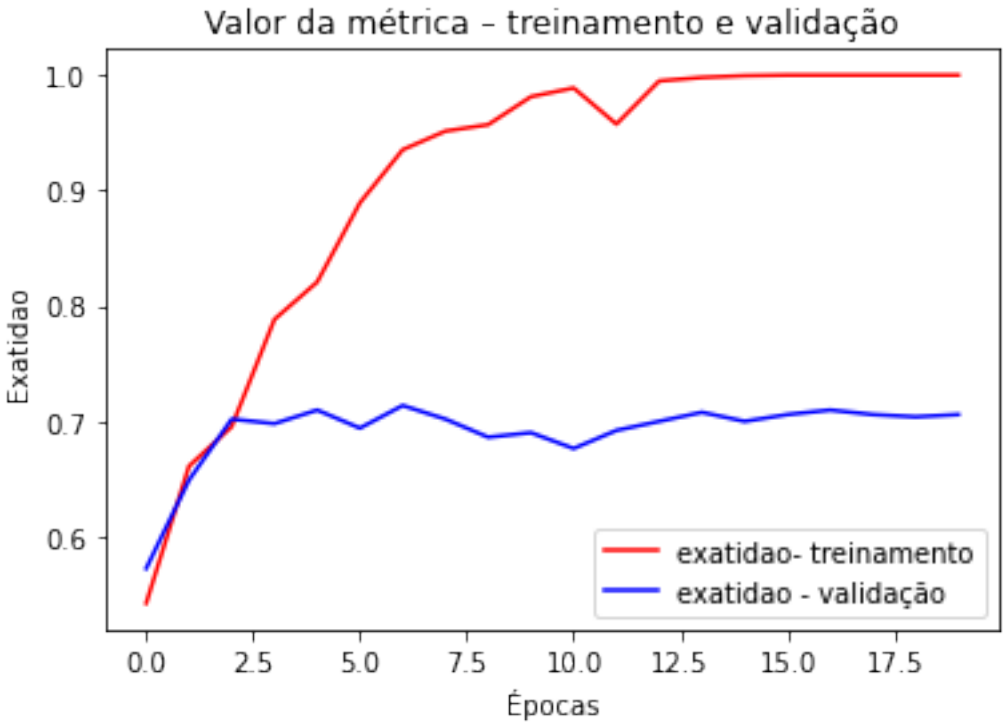
```
Epoch 1/20
66/66 - 9s - loss: 0.9156 - accuracy: 0.5429 - val_loss: 0.7954 - val_accuracy: 0.5784
Epoch 2/20
66/66 - 9s - loss: 0.6867 - accuracy: 0.6515 - val_loss: 0.6980 - val_accuracy: 0.6686
.
.
.
Epoch 19/20
66/66 - 9s - loss: 4.3007e-04 - accuracy: 1.0000 - val_loss: 2.1578 - val_accuracy: 0.6863
Epoch 20/20
66/66 - 9s - loss: 3.3048e-04 - accuracy: 1.0000 - val_loss: 2.1160 - val_accuracy: 0.7000
```

4.3 Visualização dos resultados de treinamento

Execute a célula abaixo para visualizar a função de custo e a exatidão em função do número de épocas de treinamento e verificar se o treinamento foi satisfatório.

[20]:
```
# Recupera resultados de treinamento do dicinário history
acc      = history.history['accuracy']
val_acc  = history.history['val_accuracy']
loss     = history.history['loss']
val_loss = history.history['val_loss']

# Cria vetor de épocas
epocas   = range(len(acc))

# Gráfico dos valores da função de custo
plt.plot(epocas, loss, 'r', label='Custo - treinamento')
plt.plot(epocas, val_loss, 'b', label='Custo - validação')
plt.title('Valor da função de custo - treinamento e validação')
plt.xlabel('Épocas')
plt.ylabel('Custo')
plt.legend()
plt.show()

# Gráfico dos valores da métrica
plt.plot(epocas, acc, 'r', label='exatidao- treinamento')
plt.plot(epocas, val_acc, 'b', label='exatidao - validação')
plt.title('Valor da métrica - treinamento e validação')
plt.xlabel('Épocas')
plt.ylabel('Exatidao')
plt.legend()
plt.show()
```


Valor da função de custo – treinamento e validação
Custo - treinamento
Custo - validação
Custo
Épocas
0.0
0.5
1.0
1.5
2.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Valor da métrica – treinamento e validação
exatidao- treinamento
exatidao - validação
Exatidao
Épocas
0.6
0.7
0.8
0.9
1.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

4.4 Avaliação do desempenho da RNA

Execute a célula abaixo para calcular a função de custo e a métrica de forma a avaliar o desempenho da RNA.

[21]:
```
# Calcula a função de custo e a métrica para os dados de treinamento, validação
 ↪e teste
custo_metrica_train = rna.evaluate(train_generator, steps=66)
custo_metrica_val = rna.evaluate(val_generator, steps=17)
custo_metrica_test = rna.evaluate(test_generator, steps=17)

print(custo_metrica_train)
print(custo_metrica_val)
print(custo_metrica_test)
```

```
66/66 [==============================] - 7s 100ms/step - loss: 2.0297e-04 -
accuracy: 1.0000
17/17 [==============================] - 2s 96ms/step - loss: 2.0621 - accuracy:
0.7059
17/17 [==============================] - 2s 94ms/step - loss: 2.0256 - accuracy:
0.7157
[0.00020297023002058268, 1.0]
[2.062084913253784, 0.7058823704719543]
[2.0256030559539795, 0.7156862616539001]
```

Saída esperada:

```
[0.0025995231699198484, 1.0]
[2.1967246532440186, 0.6921568512916565]
[2.334252119064331, 0.6980392336845398]
```

4.5 Análise dos resultados

Como você pode observar essa RNA está com problema de "overfitting". Isso pode ser observado de duas formas:

1) O comportamento da função de custo e da métrica para os dados de validação e para os dados de treinamento. Enquanto que a tendência da função de custo é de diminuir e da métrica (exatidão) é de aumentar para os dados de treinamento durante todo o treinamento, para os dados de validação após algumas épocas, a função de custo aumenta e a exatidão diminui. Esse comportamento é tipico de problemas de "overfitting".

2) Enquanto que a exatidão para os dados de treinamento é de 100%, para os dados de validação é de apenas cerca de 70%.

3) **Importante:** Observe que não adianta treinar a RNA por um número maior de épocas que o seu desempenho não vai melhorar.

Como temos um número relativamente pequeno de exemplos de treinamento (1.980) o problema de "overfitting" é quase impossível de evitar, mas temos que eliminá-lo senão a nossa RNA não tem nenhuma utilidade. Como já visto, "overfitting" ocorre quando uma RNA é treinada com

poucos exemplos e, assim, aprende padrões que não generalizam para novos dados, ou seja, quando a RNA começa a usar caracteríticas irrelevantes presentes nos dados para fazer previsões. Por exemplo, se você, como humano, vê apenas três imagens de pessoas que são lenhadores e três imagens de pessoas que são marinheiros, e entre elas a única pessoa que usa boné é um lenhador, você pode começar a pensar que usar boné é um sinal de ser lenhador em oposição a um marinheiro. Ao fazer isso, você faria um classificador de lenhador/marinheiro muito deficiente.

Com já visto o problema de "overfitting" é um dos principais problemas do aprendizado de máquina. Dado que estamos ajustando os parâmetros de nosso modelo para um determinado conjunto de dados, como podemos garantir que as representações aprendidas pelo modelo sejam aplicáveis a dados nunca vistos antes? Como evitamos aprender coisas específicas presentes nos dados de treinamento?

Nós já vimos e testamos alguns métodos de regularização. No restante desse trabalho você vai usar geração artifical de dados ("data augmentation") para tentar minimizar o problema de "overfitting" dessa RNA.

4.6 Teste da RNA com novas imagens

O código da célula abaixo permite que você escolher um ou mais arquivos que estão no seu computador, carregar esses arquivos e utilizar a sua RNA para prever se a imagem carregada mostra um gato, ou um cão, ou um panda. Observa-se que esse código somente funciona se você estiver utilizando o Colab.

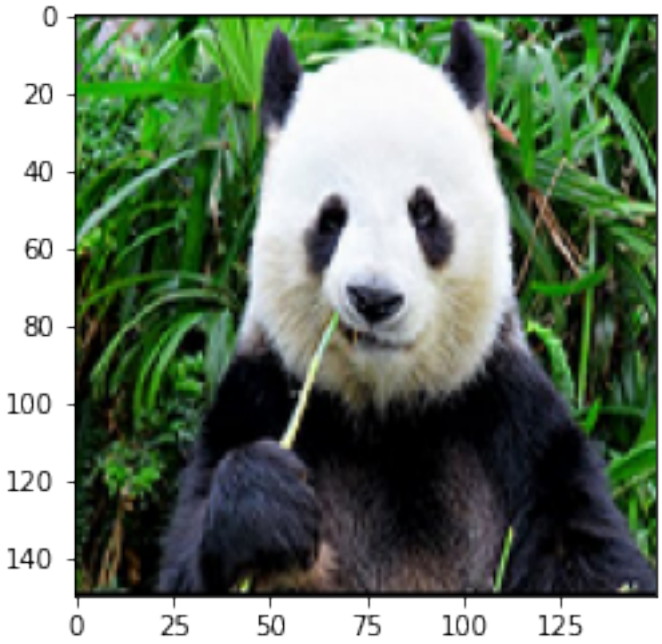
Observação. Um bom site da internet para obter imagens em geral é o pixabay (https://pixabay.com/pt/).

[22]:
```
# Importa bibliotecas e funções
import numpy as np
from google.colab import files
from keras.preprocessing import image

# Função do Colab para carregar arquivos
uploaded = files.upload()

for fn in uploaded.keys():
    path = '/content/' + fn
    # Carrega imagens usando função do Keras
    img = image.load_img(path, target_size=(150, 150))

    # Mostra imagem
    plt.imshow(img)
    plt.show()

    # Converte imagem para tensor e acrescenta eixo dos exemplos
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    images = np.vstack([x])
```

```
    # Calcula previsão da RNA e determina classe
    y_prev = rna.predict(images, batch_size=10)
    classe = np.argmax(y_prev)

    # Apresenta classe identificada
    if classe==0:
        print(fn + " é um gato")
    elif classe==1:
        print(fn + " é um cão")
    else:
        print(fn + " é um panda")
```

```
<IPython.core.display.HTML object>
```

```
Saving panda.jpg to panda (1).jpg
```




```
panda.jpg é um panda
```

5 Treinamento com geração artificial de dados

Para treinar a sua RNA com "data augmentation" você precisa de um gerador de dados de treinamento que além de carregar as imagens, normalizá-las, redimensioná-las e convertê-las em tensores de números reais, deve também modificá-las aplicando transformações aleatórias. Isso perimte que a RNA seja treinada sempre com novas imagens, mas obviamente que são imagens criadas artificialmente baseadas nas imagens originais de treinamento.

Como não se deve realizar transformações das imagens de validação e teste, podemos utilizar para carregar e pré-processar essas imagens os mesmos geradores que criamos anteriormente, ou seja, `val_generator` e `test_generator`. Dessa forma precisamos criar e instanciar somente um novo gerador para as imagens de treinamento.

5.1 Exercício #6: Gerador de dados de treinamento com "data augmentation"

Para criar e configurar o gerador de imagens de treinamento com "data augmentation" você deve usar o mesmo procedimento realizado no exercício #3. Esse gerador devem ser configurado e instanciado da seguinte forma:

- Normalização dos pixles para valores no intervalo [0, 1]
- Tamanho do lote = 30
- Tipo de problema: classificação multiclasse
- Dimensão das imagens: 150x150
- Redução/ampliação de 20%
- Angulo de rotação no intervalo de +-20 graus
- Translação vertical e horizontal de 0,2
- Inversão horizontal
- Alteração do brilho entre 0,5 e 1,1
- Método de preenchimento de lacunas: borda

[23]:
```
# PARA VOCÊ FAZER: compilar e treinar a RNA

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define dimensão das imagens
img_size_gen = (150, 150)

# Criação do gerador de dados de treinamento com "data augmentation"
# Incluir seu código aqui
aug_datagen = ImageDataGenerator(rescale=1/255, brightness_range=[0.5, 1.1],␣
 ↪fill_mode='wrap',
                                 horizontal_flip=True, zoom_range=.2,␣
 ↪rotation_range=20)

# Instancia gerador de imagens de treinamento (utilize a variável que define o␣
 ↪diretório de dados de treinamento)
# Incluir seu código aqui
aug_train_generator = aug_datagen.flow_from_directory(os.path.join(base_dir,␣
 ↪'train'), target_size=img_dim,class_mode='categorical',batch_size=30)
```

```
Found 1980 images belonging to 3 classes.
```

Saída esperada:

```
Found 1980 images belonging to 3 classes.
```

5.2 Exercício #7: Teste do gerador com "data augmentation"

Modifique a célula abaixo para executar o gerador de imagens com "data augmentation" de forma a criar um lote de imagens transformadas. Utilize para isso o método `next()`.

[25]:
```
# PARA VOCÊ FAZER: testar gerador com "data augmentation"

import numpy as np

# Inicializa tensores de imagens e saídas
imagens, y = [], []

# Executa o gerador uma única vez
# Incluir seu código aqui
imagens, y = aug_train_generator.next()


# Tranforma a saída da rede que é uma probabilidade de mostrar um dos animais na␣
 ↪classe prevista.
# Para isso use a função argmax com axis=1.
# Incluir seu código aqui
#
classes = np.argmax(rna.predict(imagens), axis=1)
# Apresenta as classes previstas
print('Img No. - Classe (one_hot) - Classe (inteiro)')
for i in range(classes.shape[0]):
    print(i, y[i], classes[i])
```

```
Img No. - Classe (one_hot) - Classe (inteiro)
0 [1. 0. 0.] 0
1 [1. 0. 0.] 0
2 [1. 0. 0.] 1
3 [0. 0. 1.] 2
4 [0. 0. 1.] 2
5 [0. 1. 0.] 0
6 [1. 0. 0.] 0
7 [0. 1. 0.] 0
8 [1. 0. 0.] 0
9 [0. 1. 0.] 1
10 [0. 1. 0.] 1
11 [0. 1. 0.] 0
12 [1. 0. 0.] 0
13 [0. 1. 0.] 0
14 [0. 1. 0.] 1
15 [0. 0. 1.] 2
16 [0. 0. 1.] 2
17 [0. 0. 1.] 2
18 [0. 1. 0.] 0
```

```
19 [0. 1. 0.] 1
20 [0. 0. 1.] 2
21 [1. 0. 0.] 0
22 [0. 0. 1.] 2
23 [1. 0. 0.] 0
24 [1. 0. 0.] 1
25 [0. 0. 1.] 0
26 [0. 0. 1.] 2
27 [0. 1. 0.] 1
28 [0. 0. 1.] 2
29 [1. 0. 0.] 1
```

Saída esperada:

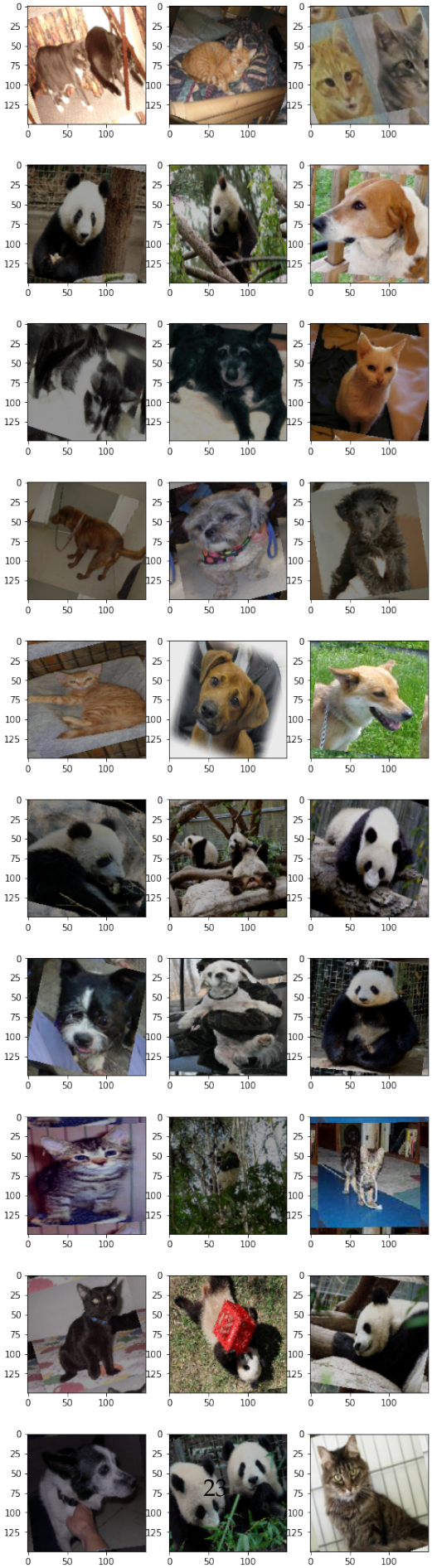
```
Img No. - Classe (one_hot) - Classe (inteiro)
0 [0. 1. 0.] 1
1 [1. 0. 0.] 0
2 [1. 0. 0.] 0
3 [1. 0. 0.] 0
4 [0. 0. 1.] 2
5 [0. 0. 1.] 2
6 [1. 0. 0.] 0
7 [0. 1. 0.] 1
8 [1. 0. 0.] 0
9 [0. 1. 0.] 1
10 [0. 0. 1.] 2
11 [1. 0. 0.] 0
12 [0. 1. 0.] 1
13 [0. 0. 1.] 2
14 [1. 0. 0.] 0
15 [0. 0. 1.] 2
16 [1. 0. 0.] 0
17 [1. 0. 0.] 0
18 [0. 0. 1.] 2
19 [0. 0. 1.] 2
20 [1. 0. 0.] 0
21 [0. 0. 1.] 2
22 [0. 1. 0.] 1
23 [1. 0. 0.] 0
24 [0. 0. 1.] 2
25 [0. 0. 1.] 2
26 [0. 1. 0.] 1
27 [0. 0. 1.] 2
28 [0. 1. 0.] 1
29 [0. 0. 1.] 2
```

5.3 Visualização das imagens transformadas

Execute a célula abaixo para visualizar o lote de imagens transformadas.

[26]:
```
fig = plt.figure(figsize=(8, 32))

nrows = 10
ncols = 3

print('Classes das imagens =', classes)

i = 0
for img in imagens:
# Set up subplot; subplot indices start at 1
    plt.subplot(nrows, ncols, i + 1)
    plt.imshow(img)
    i = i + 1
plt.show()
```

```
Classes das imagens = [0 0 1 2 2 0 0 0 0 1 1 0 0 0 1 2 2 2 0 1 2 0 2 0 1 0 2 1 2
1]
```



5.4 Exercício #8: Compilação e treinamento da RNA

Agora você vai treinar a sua RNA usando o método de otimização Adams e o gerador de dados com "data augmentation". Assim, na célula abaixo, crie uma nova RNA, compile e treine usando os seguintes hiperparâmetros:

- método de otimização: Adam;
- taxa de aprendizagem = 0.001;
- número de épocas = 30;
- verbose = 2.

Para esse treinamento com "data augmentation" oberve os seguintes pontos:

- Cuidado para definir os parâmetros `steps_per_epoch` e `validation_steps` de forma a utilizar todos as imagens de treinamento e de validação. Lembre que temos 1980 imagens de treinamento, 510 imagens de validação e o tamanho dos lotes é de 30 imagens.
- Para treinar a RNA, utilize o método `fit_generator`, o gerador de dados de treinamento com "data augmentation, e o gerador de dados de validação definidos anteriormente.
- Utilize a mesma função de custo e a mesma métrica usadas anteriormentes.

Importante: Com "data augmentation" temos que treinar a RNA por um número maior de épocas para que ela seja capaz de generalizar os dados de treinamento de forma adequada.

[28]:
```
# PARA VOCÊ FAZER: Criar, compilar e treinar a RNA com "data augmentation"

from tensorflow.keras import optimizers

# Cria RNA usando a função build_model
# Incluir seu código aqui
rna2 = build_model(img_size)

# Comiplação da RNA
# Incluir seu código aqui
rna2.compile(opt, loss='categorical_crossentropy', metrics='accuracy')

# Treinamento da RNA
# Incluir seu código aqui
history = rna2.fit_generator(aug_train_generator, steps_per_epoch=66, epochs=30,␣
 ↪verbose=2, validation_data=val_generator, validation_steps=17)
```

```
WARNING:tensorflow:From <ipython-input-28-284b7f30ba39>:15: Model.fit_generator
(from tensorflow.python.keras.engine.training) is deprecated and will be removed
in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/30
66/66 - 18s - loss: 1.5291 - accuracy: 0.3702 - val_loss: 0.9493 - val_accuracy:
```

```
0.4824
Epoch 2/30
66/66 - 18s - loss: 0.9405 - accuracy: 0.5611 - val_loss: 0.8785 - val_accuracy:
0.5627
Epoch 3/30
66/66 - 18s - loss: 0.8471 - accuracy: 0.5828 - val_loss: 0.9102 - val_accuracy:
0.5020
Epoch 4/30
66/66 - 18s - loss: 0.8322 - accuracy: 0.5838 - val_loss: 0.7746 - val_accuracy:
0.5863
Epoch 5/30
66/66 - 18s - loss: 0.7922 - accuracy: 0.6005 - val_loss: 0.7761 - val_accuracy:
0.6275
Epoch 6/30
66/66 - 18s - loss: 0.7680 - accuracy: 0.6096 - val_loss: 0.8085 - val_accuracy:
0.6255
Epoch 7/30
66/66 - 18s - loss: 0.7450 - accuracy: 0.6232 - val_loss: 0.7737 - val_accuracy:
0.6059
Epoch 8/30
66/66 - 18s - loss: 0.7458 - accuracy: 0.6197 - val_loss: 0.7337 - val_accuracy:
0.6333
Epoch 9/30
66/66 - 18s - loss: 0.7121 - accuracy: 0.6525 - val_loss: 0.7719 - val_accuracy:
0.6333
Epoch 10/30
66/66 - 18s - loss: 0.6980 - accuracy: 0.6652 - val_loss: 0.7679 - val_accuracy:
0.6471
Epoch 11/30
66/66 - 18s - loss: 0.6768 - accuracy: 0.6697 - val_loss: 0.6977 - val_accuracy:
0.6510
Epoch 12/30
66/66 - 18s - loss: 0.6600 - accuracy: 0.6793 - val_loss: 0.7758 - val_accuracy:
0.6392
Epoch 13/30
66/66 - 18s - loss: 0.6529 - accuracy: 0.6859 - val_loss: 0.7220 - val_accuracy:
0.6588
Epoch 14/30
66/66 - 18s - loss: 0.6364 - accuracy: 0.6899 - val_loss: 0.6629 - val_accuracy:
0.6765
Epoch 15/30
66/66 - 18s - loss: 0.6148 - accuracy: 0.7005 - val_loss: 0.6946 - val_accuracy:
0.6922
Epoch 16/30
66/66 - 18s - loss: 0.5972 - accuracy: 0.7106 - val_loss: 0.6831 - val_accuracy:
0.6863
Epoch 17/30
66/66 - 18s - loss: 0.5920 - accuracy: 0.7207 - val_loss: 0.7205 - val_accuracy:
```

```
0.6784
Epoch 18/30
66/66 - 18s - loss: 0.5777 - accuracy: 0.7162 - val_loss: 0.6247 - val_accuracy:
0.7118
Epoch 19/30
66/66 - 18s - loss: 0.5543 - accuracy: 0.7318 - val_loss: 0.6675 - val_accuracy:
0.6863
Epoch 20/30
66/66 - 18s - loss: 0.5509 - accuracy: 0.7364 - val_loss: 0.6748 - val_accuracy:
0.6882
Epoch 21/30
66/66 - 18s - loss: 0.5389 - accuracy: 0.7434 - val_loss: 0.7424 - val_accuracy:
0.6667
Epoch 22/30
66/66 - 18s - loss: 0.5297 - accuracy: 0.7540 - val_loss: 0.8720 - val_accuracy:
0.6294
Epoch 23/30
66/66 - 18s - loss: 0.5151 - accuracy: 0.7621 - val_loss: 0.6574 - val_accuracy:
0.7176
Epoch 24/30
66/66 - 18s - loss: 0.5041 - accuracy: 0.7747 - val_loss: 0.6557 - val_accuracy:
0.7275
Epoch 25/30
66/66 - 18s - loss: 0.5066 - accuracy: 0.7702 - val_loss: 0.6473 - val_accuracy:
0.7118
Epoch 26/30
66/66 - 18s - loss: 0.4716 - accuracy: 0.7828 - val_loss: 0.6586 - val_accuracy:
0.7157
Epoch 27/30
66/66 - 18s - loss: 0.4546 - accuracy: 0.7934 - val_loss: 0.6479 - val_accuracy:
0.7255
Epoch 28/30
66/66 - 18s - loss: 0.4545 - accuracy: 0.7934 - val_loss: 0.6106 - val_accuracy:
0.7314
Epoch 29/30
66/66 - 18s - loss: 0.4416 - accuracy: 0.8035 - val_loss: 0.6776 - val_accuracy:
0.7118
Epoch 30/30
66/66 - 18s - loss: 0.4445 - accuracy: 0.8116 - val_loss: 0.6308 - val_accuracy:
0.7157
```

Saída esperada:

```
Epoch 1/30
66/66 - 19s - loss: 0.9874 - accuracy: 0.4939 - val_loss: 0.7968 - val_accuracy: 0.5843
Epoch 2/30
66/66 - 19s - loss: 0.8382 - accuracy: 0.5823 - val_loss: 0.7492 - val_accuracy: 0.5843
Epoch 3/30
.
```

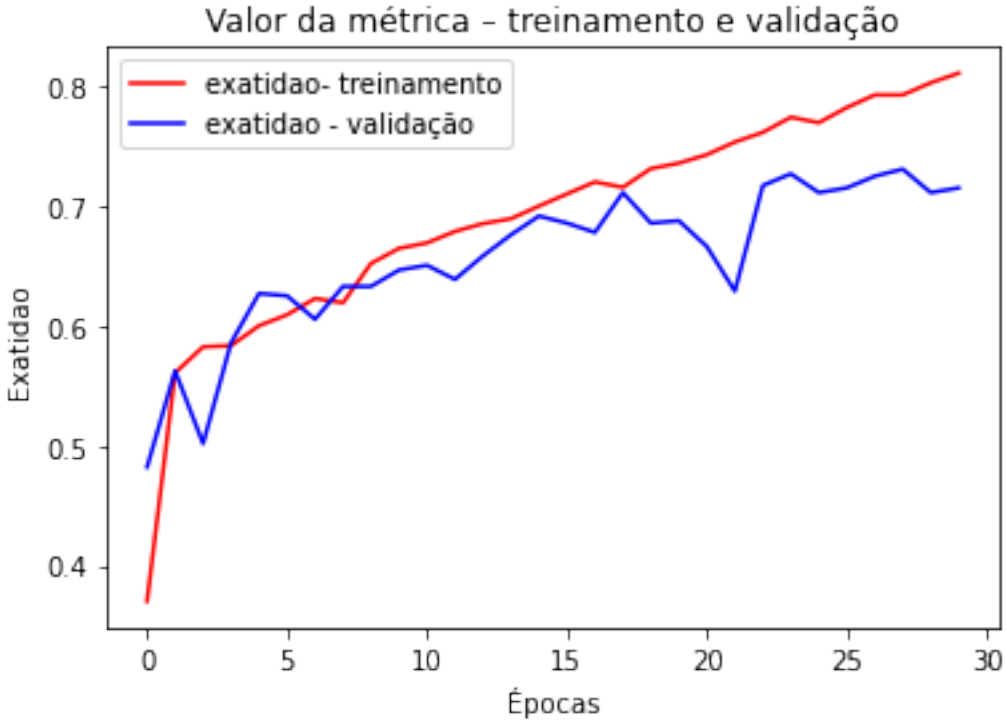
```
.
.
Epoch 29/30
66/66 - 19s - loss: 0.4986 - accuracy: 0.7566 - val_loss: 0.6341 - val_accuracy: 0.7824
Epoch 30/30
66/66 - 19s - loss: 0.4823 - accuracy: 0.7773 - val_loss: 0.6664 - val_accuracy: 0.7745
```

5.5 Visualização dos resultados de treinamento

Execute a célula abaixo para visualizar a função de custo e a exatidão em função do número de épocas de treinamento e verificar se o treinamento foi satisfatório.

[29]:
```
# Recupera resultados de treinamento do dicinário history
acc      = history.history['accuracy']
val_acc  = history.history['val_accuracy']
loss     = history.history['loss']
val_loss = history.history['val_loss']

# Cria vetor de épocas
epocas   = range(len(acc))

# Gráfico dos valores da função de custo
plt.plot(epocas, loss, 'r', label='Custo - treinamento')
plt.plot(epocas, val_loss, 'b', label='Custo - validação')
plt.title('Valor da função de custo - treinamento e validação')
plt.xlabel('Épocas')
plt.ylabel('Custo')
plt.legend()
plt.show()

# Gráfico dos valores da métrica
plt.plot(epocas, acc, 'r', label='exatidao- treinamento')
plt.plot(epocas, val_acc, 'b', label='exatidao - validação')
plt.title('Valor da métrica - treinamento e validação')
plt.xlabel('Épocas')
plt.ylabel('Exatidao')
plt.legend()
plt.show()
```


Valor da função de custo – treinamento e validação
Custo - treinamento
Custo - validação
Custo
Épocas
1.4
1.2
1.0
0.8
0.6
0.4
0
5
10
15
20
25
30


Valor da métrica – treinamento e validação
exatidao- treinamento
exatidao - validação
Exatidao
Épocas
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25
30

5.6 Avaliação do desempenho da RNA

Execute a célula abaixo para calcular a função de custo e a métrica de forma a avaliar o desempenho da RNA.

[30]:
```
# Calcula a função de custo e a métrica para os dados de treinamento, validação
↪e teste
custo_metrica_train = rna2.evaluate(train_generator, steps=66)
custo_metrica_val = rna2.evaluate(val_generator, steps=17)
custo_metrica_test = rna2.evaluate(test_generator, steps=17)

print(custo_metrica_train)
print(custo_metrica_val)
print(custo_metrica_test)
```

```
66/66 [==============================] - 7s 99ms/step - loss: 0.4001 - accuracy:
0.8242
17/17 [==============================] - 2s 96ms/step - loss: 0.6308 - accuracy:
0.7157
17/17 [==============================] - 2s 95ms/step - loss: 0.6959 - accuracy:
0.6804
[0.4000893235206604, 0.8242424130439758]
[0.63084477186203, 0.7156862616539001]
[0.6958518028259277, 0.6803921461105347]
```

Saída esperada:

```
[0.38780757784843445, 0.8287878632545471]
[0.5447821617126465, 0.7549019455909729]
[0.48929181694984436, 0.7862744927406311]
```

5.7 Análise dos resultados

Como você pode observar essa RNA treinada com "data augmetation" apresenta muito menos problema de "overfitting". Isso pode ser observado pelo seguinte:

1) O comportamento da função de custo e da métrica para os dados de validação não divergem tanto em relação aos valores dos dados de treinamento, como era no caso sem "data augmentation".

2) Os valores das métrica para os dados de validação e de teste são mais próximos dos valores obtidos para os dados de treinamento.

O desempenho da RNA para as imagens de teste aumentou cerca de 5%, o que pode ser considerado um bom resultado tendo em vista que não tivemos quase nenhum trabalho para produzir novos exemplos de terinamento.

5.8 Teste da RNA com novas imagens

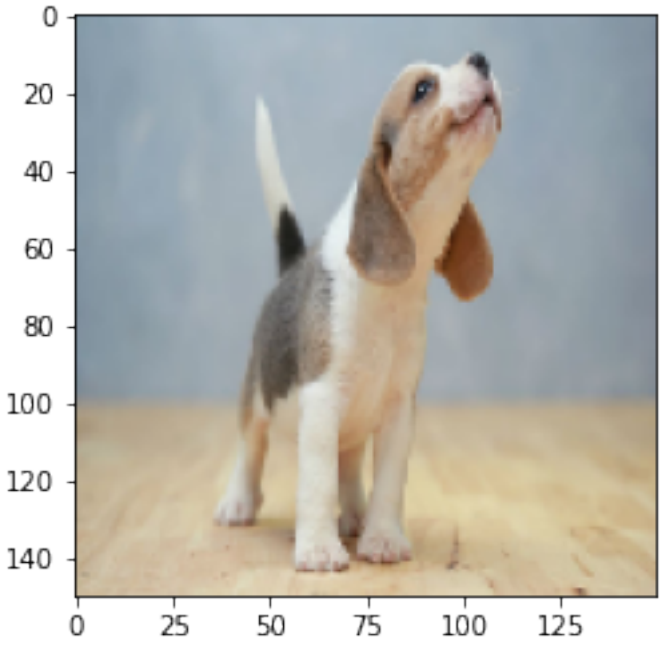
Execute a célula abaixo para escolher um ou mais arquivos que estão no seu computador, carregar esses arquivos e utilizar a sua RNA com "data augmentation" para prever se a imagem carregada

mostra um gato, ou um cão, ou um panda. Observa-se que esse código somente funciona se você estiver utilizando o Colab.

[31]:
```
# Importa bibliotecas e funções
import numpy as np
from google.colab import files
from keras.preprocessing import image

# Função do Colab para carregar arquivos
uploaded = files.upload()

for fn in uploaded.keys():
    path = '/content/' + fn
    # Carrega imagens usando função do Keras
    img = image.load_img(path, target_size=(150, 150))

    # Mostra imagem
    plt.imshow(img)
    plt.show()

    # Converte imagem para tensor e acrescenta eixo dos exemplos
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    images = np.vstack([x])

    # Calcula previsão da RNA e determina classe
    y_prev = rna.predict(images, batch_size=10)
    classe = np.argmax(y_prev)

    # Apresenta classe identificada
    if classe==0:
        print(fn + " é um gato")
    elif classe==1:
        print(fn + " é um cão")
    else:
        print(fn + " é um panda")
```

```
<IPython.core.display.HTML object>
```

```
Saving dog.jpg to dog (1).jpg
```




```
dog.jpg é um cão
```

5.9 Exercício #9: Responda a seguinte pergunta: Como melhorar o desempenho dessa rede?

1. Mais Data Augmentation

Usando outras estratégias de data augmentation, como random cropping e deformações elásticas podemos criar imagens novas e aumentar a base de treinamento e também a capacidade da rede neural em reconhecer a classe de interesse.

2. Melhoria no modelo

É necessário fazer um tunning da arquitetura do modelo e dos hiperparâmetros do otimizador. Uma estratégia comum é a utilização de duas ou mais camadas ocultas ao invés de uma após a camada flatten, o que não foi feito nesse exemplo.

6 Conclusão

As conclusões que podemos extrair desse trabalho são as seguintes:

1) Geração artificial de dados ("data augmentation") é uma ferramenta poderosa para minimizar problemas de "overfitting". Mas obviamente tem as suas limitações, pois os novos exemplos são criados a partir dos exemplos existentes, ou seja, não se criam de fato novos exemplos.

2) "Data augmentation" quando aplicado em conjunto com outras técnicas de regularização, tais como, regularização L2 e "dropout" tem a capacidade de eliminar "overfitting" e, assim,

obter RNAs capazes de apresentar um alto desempenho, até mesmo superior ao dos seres humanos.

7 Finalizando

Antes de sair do Colab ou inciar um outro notebook, execute a célula abaixo para finalizar o kernel e liberar memória.

[ ]:
```
import os, signal
os.kill(os.getpid(), signal.SIGKILL)
```