Trabalho #3 - Classificação binária com rede neural deep-learning

Nesse trabalho você vai desenvolver uma rede neural rasa e uma deep-learning, usando a plataforma TensorFlow-Keras, para realizar uma tarefa de classificação binária para reconhecer gatos em imagens e comparar o desempenho das duas redes.

Coloque os nomes e RAs dos alunos que fizeram esse trabalho

Nome e número dos alunos da equipe:

Aluno 1: Igor Amaral Correa 20.83992-8

Aluno 2:

1 - Bibliotecas

Em primeiro ligar, execute a célula abaixo para importar algumas bibliotecas Python que são usadas nesse trabalho.

- numpy é a biblioteca básica para computação cinetífica usando Python.
- h5py é uma biblioteca que fornece funções para interagir com banco de dados salvos em arquivos no formato h5.
- matplotlib é uma biblioteca famosa usada para fazer gráficos com Python.
- PIL e scipy são usadas para testar a sua RNA com novas imagens no final do trabalho.
- lr_utils é um arquivo com uma função para ler os dados usados nesse trabalho.

O comando da célula abaixo importa o TensorFlow e verifica a versão que está sendo utilizada.

```
import tensorflow as tf
tf.__version__
```

```
'2.2.0-rc2'
```

Como o TensorFlow é um software aberto não existe muita preocupação do desenvolvedor (no caso o Google) de manter a compatibilidade entre as versões. Assim, se você estiver usando uma versão do TensorFlow incompatível com o seu programa, você pode impor o uso de uma versão mais antiga.

Por exemplo o código abaixo importa a versão 2 do TensorFlow, mas se ela não estiver instalada importa a versão instalada.

```
try:
    import tensorflow.compat.v2 as tf
except Exception:
    import tensorflow as tf
print(tf.__version__)
```

Outro exemplo, se a versão instalada do TensorFlow for a 2 e o seu programa foi feito para a versão 1, então pode habilitar a versão 2 com o comportamento da versão 1. O código abaixo mostra como fazer isso.

```
import tensorflow as tf
print(tf.__version__)
tf.enable_v1_behavior()
```

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
from PIL import Image
from scipy import ndimage
from lr_utils import load_dataset

%matplotlib inline
```

2 - Visão geral do problema

Definição do problema:

Nesse trabalho é fornecido um conjunto de dados que contém:

```
- um conjunto de imagens de treinamento classificadas como tendo gato (y=1) ou não tendo gato (y=0);
- um conjunto de imagens de teste classificadas como tendo gato (y=1) ou não tendo gato (y=0);
- a dimensão de cada imagem é (num_px, num_px, 3), onde 3 é o número de canais de cor (RGB);
- cada imagem é composta por três matrizes de dimensão: número de linhas = num_px e número de colunas = num_px;
- número de imagens do conjunto de treinamento: m_train;
- número de imagens do conjunto de teste: m_test.
```

A partir desses conjuntos de dados, a sua tarefa é desenvolver um sistema baseado em uma rede neural, para processar imagens e classificar corretamente se ela mostra ou não um gato. Para isso, você vai desenvolver algumas redes neurais usando a ferramenta Keras e comparar o desempenho delas.

O uso da ferramenta Keras para desenvolver redes neurais facilita muito o trabalho, pois não exige desenvolver codigos específicos para implementar cada camada da rede, a função de custo, a métrica e principalmente o gradiente descendente.

3 - Conjunto de dados de treinamento e teste

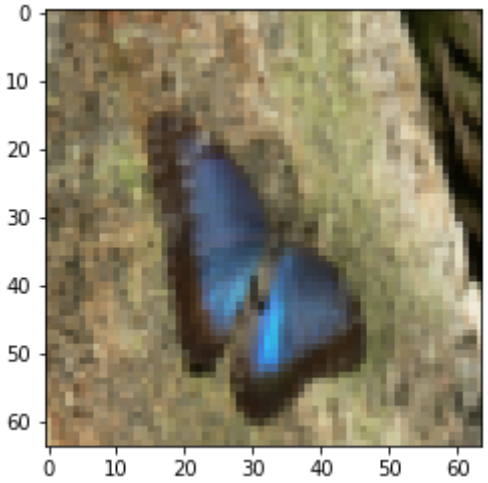
Para carregar o conjunto de dados execute o código a seguir.

O termo _orig é adicionado no final dos tensores com os dados de treinamento e teste originais porque vamos processar esses dados.

```
# Carregando os dados (gato/não-gato)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
```

```
# Examplo de uma imagem
index = 51
print(train_set_x_orig[index].shape)
plt.imshow(train_set_x_orig[index])
print ("y = " + str(train_set_y[:, index]) + ", it's a '" + classes[np.squeeze(train_set_y[:, index])].decode("utf-8") +  "' picture
```

```
(64, 64, 3)
y = [0], it's a 'non-cat' picture.
```



No código acima index é o número sequencial da imagem. Tente trocar a imagem, mudando o index, usando valores entre 0 e 208, para visualizar outros exemplos.

3.1 - Determinação do formato e dimensões dos dados

Exercício #1:

É importante conhecer as dimensões dos dados que estamos trabalhando para evitar problemas. Assim, obtenha os valores dos seguintes parâmetros: - m_train = número de exemplos de treinamento; - m_test = número de exemplos de teste; - num_px = altura e largura das imagens (as imagens são quadradas).

Lembre que `train_set_x_orig` é um tensor numpy de dimensão (m_train, num_px, num_px, 3). Por exemplo, você pode obter `m_train` escrevendo `train_set_x_orig.shape[0]`.

```
# PARA VOCÊ FAZER:

### COMECE AQUIE ### (≈ 3 linhas)
```

```
m_train = train_set_x_orig.shape[0]
m_test = test_set_x_orig.shape[0]
num_px = train_set_x_orig[0].shape[0]
### TERMINE AQUI ###

print ("Número de exemplos de treinamento: m_train = " + str(m_train))
print ("Número de exemplos de teste: m_test = " + str(m_test))
print ("Altura/largura de cada imagem: num_px = " + str(num_px))
print ("Dimensão de cada imagem: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("Dimensão - train_set_x: " + str(train_set_x_orig.shape))
print ("Dimensão - train_set_y: " + str(train_set_y.shape))
print ("Dimensão - test_set_x: " + str(test_set_x_orig.shape))
print ("Dimensão - test_set_y: " + str(test_set_y.shape))
```

```
Número de exemplos de treinamento: m_train = 209
Número de exemplos de teste: m_test = 50
Altura/largura de cada imagem: num_px = 64
Dimensão de cada imagem: (64, 64, 3)
Dimensão - train_set_x: (209, 64, 64, 3)
Dimensão - train_set_y: (1, 209)
Dimensão - test_set_x: (50, 64, 64, 3)
Dimensão - test_set_y: (1, 50)
```

Saída esperada para m_train, m_test e num_px:

Número de exemplos de treinamento: m_train = 209

Número de exemplos de teste: m_test = 50

Altura/largura de cada imagem: num_px = 64

Dimensão de cada imagem: (64, 64, 3)

Dimensão - train_set_x: (209, 64, 64, 3)

Dimensão - train_set_y: (1, 209)

Dimensão - test_set_x: (50, 64, 64, 3)

Dimensão - test_set_y: (1, 50)

Observe que o primeiro eixo dos tensores com as imagens dos conjuntos de teste e de treinamento representa os exemplos, da forma como é

3.2 - Processamento dos dados

Os dados dos exemplos de treinamento e de teste devem ser processados de forma a serem colocados em tensores com as dimensões adequadas e normalizados corretamente. Nos exercícios que seguem você irá realizar o processamento dos dados de forma a poderem ser usados na sua RNA implementada com o Keras.

Redimensionamento dos dados

Os dados de entrada de uma camada de neurônios densa é um vetor, assim, devemos redimensionar as imagens, que tem dimensão (num_px, num_px, 3), para transformá-las em um vetor linha de dimensão (1, num_px∗num_px∗3). Após esse redimensionamento o conjunto de dados é um tensor numpy onde cada linha representa uma imagem "esticada". O tensor com as entradas dos dados de treinamento terá m_train linhas e o de teste m_test linhas.

Por exemplo, para redimensionar uma matriz de dimensão (a,b,c,d) para uma matriz de dimensão (a, b∗c∗d) pode-se usar o seguinte código Python:

```
X_flatten = X.reshape((a, b∗c∗d))
```

A dimensão dos dados de saída também deve ser alterada porque, como visto, o Keras espara que os exemplos representam o primeiro eixo do tensor e nos dados de saída os exemplos representam o segundo eixo. Assim, os vetores com os dados de saída, tanto de treinamento como de teste, devem ser transpostos.

Exercício #2:

Implemente o redimensionamento dos dados de entrada na célula a seguir.

```
# PARA VOCÊ FAZER: redimensionamento das imagens dos exemplos de treinamento e teste

### COMECE AQUI ### (≈ 2 linhas)
# Redimensionamento dos dados de entrada
train_set_x_flatten = train_set_x_orig.reshape(( m_train , num_px * num_px * 3))
```

```
test_set_x_flatten = test_set_x_orig.reshape(( m_test , num_px * num_px * 3))
### TERMINE AQUI ###

### COMECE AQUI ### (≈ 2 linhas)
# Redimensionamento dos dados de saída (transposição)
train_set_y = train_set_y.reshape(( train_set_y.shape[1] , train_set_y.shape[0] ))
test_set_y = test_set_y.reshape(( test_set_y.shape[1] , test_set_y.shape[0] ))
### TERMINE AQUI ###

print ("Dimensão - train_set_x_flatten: " + str(train_set_x_flatten.shape))
print ("Dimensão - train_set_y: " + str(train_set_y.shape))
print ("Dimensão - test_set_x_flatten: " + str(test_set_x_flatten.shape))
print ("Dimensão - test_set_y: " + str(test_set_y.shape))
print ("Verificação de valores após redimensionamento: " + str(train_set_x_flatten[0,0:5]))
```

```
Dimensão - train_set_x_flatten: (209, 12288)
Dimensão - train_set_y: (209, 1)
Dimensão - test_set_x_flatten: (50, 12288)
Dimensão - test_set_y: (50, 1)
Verificação de valores após redimensionamento: [17 31 56 22 33]
```

Saída esperada:

Dimensão - train_set_x_flatten: (209, 12288)

Dimensão - train_set_y: (209, 1)

Dimensão - test_set_x_flatten: (50, 12288)

Dimensão - test_set_y: (50, 1)

Verificação de valores após redimensionamento: [17 31 56 22 33]

Normalização dos dados

Nas imagens coloridas as cores vermelho, verde e azul (RGB) são especificadas para cada pixel da imagem, de forma que cada pixel consite de um vetor de tres números que variam no intervalo de 0 a 255.

Uma etapa do processamento dos dados é normalizar esses dados de forma a ter dados com média zero e desvio padrão um. No caso de imagens o processo de normalização dos dados é mais simples e consiste simplesmente em dividir todos os valores da imagem por 255, que é o valor máximo de um pixel.

Exercício #3:

```
# PARA VOCÊ FAZER: redimensionamento das imagens dos exemplos de treinamento e teste

### COMECE AQUI ### (≈ 2 linhas)
train_set_x = train_set_x_flatten / 255
test_set_x = test_set_x_flatten / 255
### TERMINE AQUI ###

print ("Máximo valor de train_set_x: " + str(np.max(train_set_x)))
print ("Máximo valor de train_set_x: " + str(np.max(test_set_x)))
```

```
Máximo valor de train_set_x: 1.0
Máximo valor de train_set_x: 1.0
```

Saída esperada:

Máximo valor de train_set_x: 1.0
Máximo valor de train_set_x: 1.0

O que é importante lembrar:

As etapas principais do processamento de dados são as seguintes:

- Verificar a dimensão e formato dos dados do problema. Nesse caso são: m_train, m_test, num_px, ...
- Redimensionar os conjuntos de dados. Nesse caso para cada exemplo a entrada é um vetor linha de dimensão (1, num_px * num_px * 3)
- Normalizar os dados

4 - Desenvolvimento e teste da rede neural rasa

Nesse trabalho vamos testar diferentes redes neurais, variando numero de camadas, numero de neurônios nas camadas e tipo de função de ativação para obtermos uma solução que apresente resultados satisfatórios.

Conforme vimos em aula, o desenvolvimento de uma RNA com o Keras é feito segundo as seguintes etapas:

1. Definição dos dados de treinamento e de teste;
2. Configuração da RNA;
3. Compilação da RNA, que também inclui a configuração do processo de treinamento pela escolha da função de custo, do otimizador e da métrica para avaliar o desempenho;
4. Treinamento da RNA;
5. Teste e avaliação do desempenho da RNA.

Observe que a etapa de definição e preparação dos dados já foi realizada.

4.1 - Configuração da rede neural

Vamos configurar, treinar e testar uma primeira rede neural simples de uma camada intermediária usando o Keras.

Para essa rede de uma camada, o parâmetro mais importante é o número de neurônios da camada intermediária. Esse número deve ser compatível com o problema que queremos resolver e também com o número de exemplos que temos disponíveis para treinar a rede.

Em linhas gerais cada imagem é representada por 12.288 números e temos 209 imagens de treinamento. Assim, no conjunto de dados de treinamento temos cerca de 2,6 milhões de valores. Existem um princípio em aprendizado supervisionado que diz que o número de parâmetros do sistema deve ser menor do que o número de valores presente no conjunto de dados utilizado para o treinamento. Assim, a sua rede neural deve ter um número de parâmetros menor do que 2,6 milhões. Se a rede possuir mais parâmetros do que o número de valores presentes no conjunto de dados, ela simplesmenet memoriza os dados de treinamento e não funciona direito para nenhum outro caso. Veremos com detalhes esse tipo de problema, mas é importante você estar ciente disso nesse momento.

Qualquer dúvida que você possa ter de como fazer esse trabalho usando o Keras, ela pode ser sanada olhando as notas da Aula 7 - Ferramentas de Desenvolvmento.

Exercício #4:

Usando o Keras configure uma rede neural com as seguintes caracteríticas:

- uma única camada intermediária com 64 neurônios e função de ativação sigmóide;
- como queremos resolver um problema de classificação binária a camada de saída deve possuir um único neurônio e ter função de ativação sigmóide.

```
# PARA VOCÊ FAZER: configuração de uma RNA rasa

# Importar do Keras classes de modelos e camadas
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

np.random.seed(1) # inicializa gerador de números aleatórios

# Configuração da rede
### COMECE AQUI ### (≈ 3 linhas)
rna = Sequential()
rna.add(Dense(units=64, activation='sigmoid', input_dim=train_set_x_flatten.shape[1]))
rna.add(Dense(units=1, activation='sigmoid'))
### TERMINE AQUI ###

# Visualização da rede
rna.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 64)                786496    
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65        
=================================================================
Total params: 786,561
Trainable params: 786,561
Non-trainable params: 0
_________________________________________________________________
```

Saída esperada:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 64)                786496    
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65        
=================================================================
Total params: 786,561
Trainable params: 786,561
Non-trainable params: 0
_________________________________________________________________
```

4.2 - Compilação e treinamento da rede neural

A segunda e terceira etapas de desenvolvimento da rede no Keras é a sua compilação e treinamento.

Exercício #5:

Compile e treine a sua rede neural usando as seguintes opções:

- Método de otimização: gradiente descendente;
- Função de custo: entropia cruzada;
- Métrica: exatidão;
- No treinamento, utilize no método fit o parâmetro batch_size=209, que é o número de exemplos de treinamento;
- Número de épocas: 10.

Nesse momento estamos treinando a rede somente para verificar se ela é adequada para resolver o problema e se está configurada de forma correta, por isso usamos poucas épocas de treinamento.

```
# PARA VOCÊ FAZER: configuração do otimizador e treinamento da RNA rasa

# Importar do Keras classe de otimizadores
from tensorflow.keras import optimizers

# Compilação da rede
### COMECE AQUI ### (≈ 1 ou 2 linhas)
rna.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['acc'])
### TERMINE AQUI ###

# Teste de treinamento da rede
### COMECE AQUI ### (≈ 1 linha)
rna.fit(train_set_x, train_set_y, epochs=10, batch_size=209)
### TERMINE AQUI ###
```

```
Epoch 1/10
1/1 [==============================] - 0s 2ms/step - loss: 0.7668 - acc: 0.3445
Epoch 2/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6749 - acc: 0.6172
Epoch 3/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6574 - acc: 0.6507
Epoch 4/10
1/1 [==============================] - 0s 1ms/step - loss: 0.6530 - acc: 0.6555
Epoch 5/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6513 - acc: 0.6555
Epoch 6/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6501 - acc: 0.6555
Epoch 7/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6490 - acc: 0.6555
Epoch 8/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6480 - acc: 0.6555
Epoch 9/10
1/1 [==============================] - 0s 1ms/step - loss: 0.6470 - acc: 0.6555
Epoch 10/10
1/1 [==============================] - 0s 3ms/step - loss: 0.6461 - acc: 0.6555
<tensorflow.python.keras.callbacks.History at 0x7f2f50174c88>
```

Saída esperada:

```
Epoch 1/10
209/209 [==============================] - 0s 1ms/sample - loss: 0.7257 - acc: 0.3876
Epoch 2/10
209/209 [==============================] - 0s 272us/sample - loss: 0.6633 - acc: 0.6555
Epoch 3/10
209/209 [==============================] - 0s 162us/sample - loss: 0.6539 - acc: 0.6459
Epoch 4/10
209/209 [==============================] - 0s 153us/sample - loss: 0.6515 - acc: 0.6507
Epoch 5/10
209/209 [==============================] - 0s 153us/sample - loss: 0.6502 - acc: 0.6507
Epoch 6/10
209/209 [==============================] - 0s 143us/sample - loss: 0.6491 - acc: 0.6507
Epoch 7/10
209/209 [==============================] - 0s 153us/sample - loss: 0.6481 - acc: 0.6507
Epoch 8/10
209/209 [==============================] - 0s 166us/sample - loss: 0.6471 - acc: 0.6507
Epoch 9/10
209/209 [==============================] - 0s 157us/sample - loss: 0.6461 - acc: 0.6507
Epoch 10/10
209/209 [==============================] - 0s 167us/sample - loss: 0.6451 - acc: 0.6507
```

Se o seu resultado é esse ou parecido com esse, você pode conluir que a rede está correta e é capaz de aprender os dados de treinamento. Assim, agora você está pronto para treinar a rede de verdade.

Exercício #6:

Retreine a sua rede usando 1000 épocas.

Use a opção de **não imprimir** os resultados parciais do treinamento para não gerar tantos dados. No Keras podemos escolher como monitoramos o progresso do treinamento com o parâmetro verbose do método fit, que nos dá as seguintes opções:

- verbose: inteiro = 0, 1, or 2.
- verbose = 0: silencioso;
- verbose = 1; barra de progresso (padrão);
- verbose = 2; uma linha por época.

Guarde os resultados do treinamento para poder fazer um gráfico do processo de treinamento. Se tiver dúvidas, veja nas notas da Aula 7 como fazer isso.

Não se esqueça de usar no método fit o parâmetro batch_size=209.

```
# PARA VOCÊ FAZER: treinamento da RNA por 1000 épocas

### COMECE AQUI ### (≈ 1 linha)
history = rna.fit(train_set_x, train_set_y, epochs=1000, batch_size=209, verbose=0)
### TERMINE AQUI ###


# Vamos verificar quais variáveis foram salvas no processo de treinamento
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'acc'])
```

4.3 - Visualização do resultado do treinamento

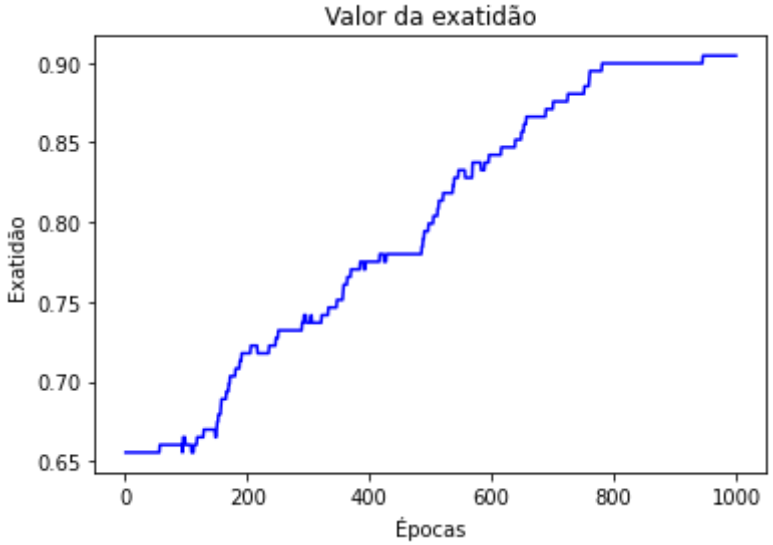
Para sabermos como o treinamento foi realizado precisamos visualizar a função de custo e a métrica ao longo do processo de treinamento.

Exercício #7:

Implemente na célula abaixo a visualização dos resultados do treinamento. Observe que os valores da função de custo e da métrico estão no dicionário history com nomes `'loss'` e `'acc'`.

Consulte as notas da Aula 7 para relembrar como fazer os gráficos da função de custo e da métrica em função do número de épocas.

```
# PARA VOCÊ FAZER: visualização do resultado do treinamento

# Salva custo e exatidão em vetores
### COMECE AQUI ### (≈ 2 linhas)
custo = history_dict['loss']
exatidao = history_dict['acc']
### TERMINE AQUI ###

# Cria vetor de épocas
### COMECE AQUI ### (≈ 1 linha)
epocas = range(1, len(custo) + 1)
### TERMINE AQUI ###

# Gráfico do custo em funçaõ das épocas
### COMECE AQUI ### (≈ 5 linhas)
plt.plot(epocas, custo, 'b')
plt.title('Valor da função de custo')
plt.xlabel('Épocas')
plt.ylabel('Custo')
plt.show()
### TERMINE AQUI ###

# Gráfico da exatidão em função das épocas
### COMECE AQUI ### (≈ 5 linhas)
plt.plot(epocas, exatidao, 'b')
plt.title('Valor da exatidão')
plt.xlabel('Épocas')
plt.ylabel('Exatidão')
plt.show()
### TERMINE AQUI ###
```



Saída esperada:

O comportamento esperado para a função de custo durante o treinamento é ela decrescer monotomicamente do início até o final. Para a exatidão, o comportamento esperado é ela aumentar durante o treinamento, apresentando algumas oscilações e alcançando um valor alto no

final.

Se você obteve esses resultados, então, a sua rede foi treinada de forma satisfatória e pode-se concluir que ela tem uma capacidade alta para se ajustar aos dados de treinamento.

4.4 - Avaliação do desempenho da rede neural

Após treinar a RNA é importante avaliar o seu desempenho com dados que não foram utilizados no treinamento. Para isso usamos o dados de teste, que estão nos tensores test_set_x e test_set_y.

Exercício #8:

Avalie a sua rede neural calculando os valores da função de custo e da exatidão para os dados de teste usando o método evaluate, conforme visto na Aula 7 - Ferramentas de desenvolvimento. Calcule esses valores também para os dados de treinamento para poder fazer comparação.

```
# PARA VOCÊ FAZER: calculo do custo e exatidão para os dados de teste

# Usando método evaluate calcule o custo e a exatidão para os dados de treinamento e depois apresente os resultados
### COMECE AQUI ### (≈ 2 linhas)
custo_e_metricas_train = rna.evaluate(train_set_x, train_set_y, batch_size=1)
print(custo_e_metricas_train)
### TERMINE AQUI ###

# Usando método evaluate calcule o custo e a exatidão para os dados de teste e depois apresente os resultados
### COMECE AQUI ### (≈ 2 linhas)
custo_e_metricas_test = rna.evaluate(test_set_x, test_set_y, batch_size=1)
print(custo_e_metricas_test)
### TERMINE AQUI ###
```

```
209/209 [==============================] - 0s 2ms/step - loss: 0.3313 - acc: 0.9043
[0.3312949240207672, 0.9043062329292297]
50/50 [==============================] - 0s 2ms/step - loss: 0.6080 - acc: 0.6800
[0.6079809069633484, 0.6800000071525574]
```

Resultados esperados:

209/209 [==============================] - 0s 656us/sample - loss: 0.3270 - acc: 0.9091
[0.3269858589868226, 0.90909094]
50/50 [==============================] - 0s 260us/sample - loss: 0.5960 - acc: 0.7000
[0.5959848165512085, 0.7]

Comentários:

- A exatidação obtida com os dados de treinamento é cerca de 90%, isso significa que a sua rede é satisfatória para resolver esse problema.
- A exatidão obtida com os dados de teste é de cerca de 68%. Esse resultado de fato não é muito bom para essa tarefa simples de classificação.

Analisando esses dados surge uma dúvida. Porque a rede não foi capaz de apresentar um bom desempenho nos dados de teste, se os resultados foram bons nos dados de treinamento?

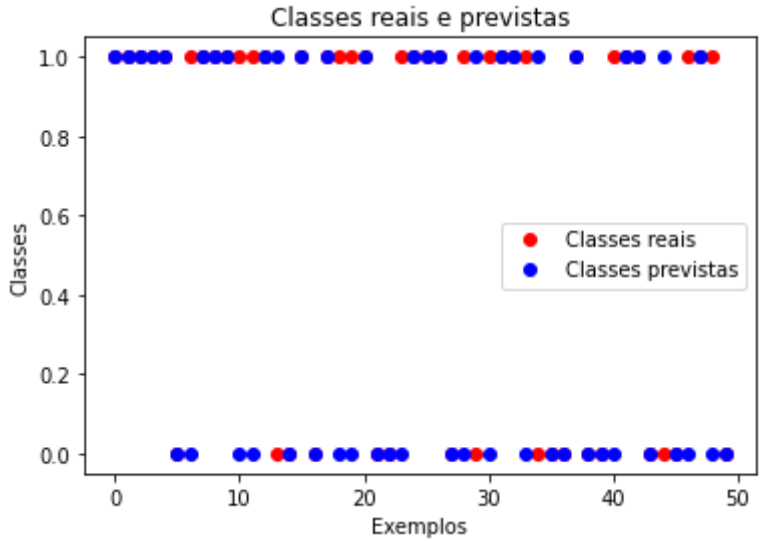
Exercício #9:

Para avaliar melhor o desepenho da sua rede calcule as saídas previstas dos exemplos do conjunto de teste usando o método predict e a função numpy round faça um gráfico com as classes reais e previstas dos dados do conjunto de teste. Se não souber como fazer consulte a Aula 7 - Ferramentas de desenvolvimento.

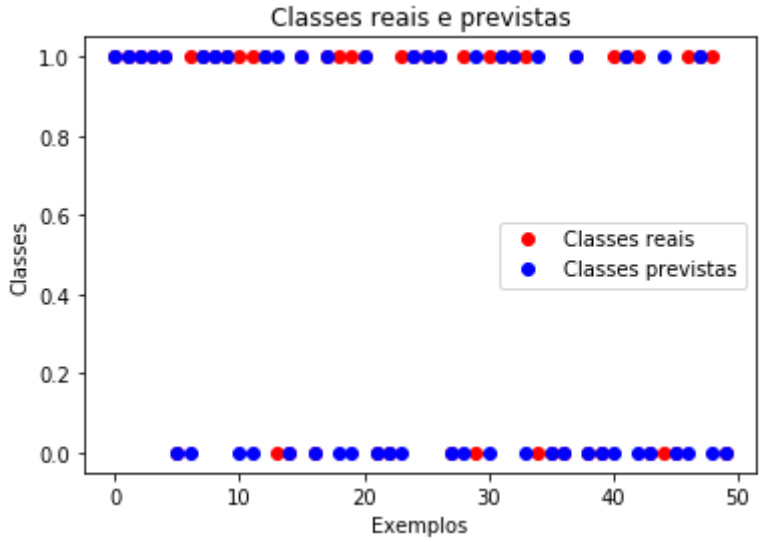
```
# PARA VOCÊ FAZER: calculo das classes previstas dos dados de teste

# Usando método predict calcule as classes previstas
### COMECE AQUI ### (≈ 2 linhas)
y_prev = rna.predict(test_set_x, batch_size=1)
yy_prev = np.round(y_prev)
### TERMINE AQUI ###

# Transforma saida prevista em números inteiros
yy_prev = yy_prev.astype(int)

# Gráfico das classes reais e previstas
```

```
# Gráfico das classes reais e previstas
### COMECE AQUI ### (≈ 7 linhas)
plt.plot(test_set_y, 'ro', label='Classes reais')
plt.plot(yy_prev, 'bo', label='Classes previstas')
plt.title('Classes reais e previstas')
plt.xlabel('Exemplos')
plt.ylabel('Classes')
plt.legend()
plt.show
### TERMINE AQUI ###
```

```
<function matplotlib.pyplot.show>
```



Saída esperada:



Importante:

No Google Colab o comando `<img src="imagem">` para mostrar uma imagem em uma célula de texto não funciona.

Para mostrar uma imagem em uma célula de texto (markdown) no **Google Colab** deve-se usar o seguinte procedimento:

1. Na célula de texto clicar em inserir imagem. Ao fazer isso o comando `![alt text](https://)` é inserido.
2. A imagem deve estar no Google Drive. Vá até onde está o arquivo da imagem no Google Drive, clique com o botão direito do mouse e selecione "Gerar link compartilhável" (em inglês "Get shareable link") e copie o link no lugar de "https://".
3. O link copiado é algo do tipo: https://drive.google.com/open?id=1GWk0IjW-SQcwQvhsgOXpjscbROdxXCUI
4. Trocar a palavra "open" por "uc" e pronto a imagem irá aparecer.

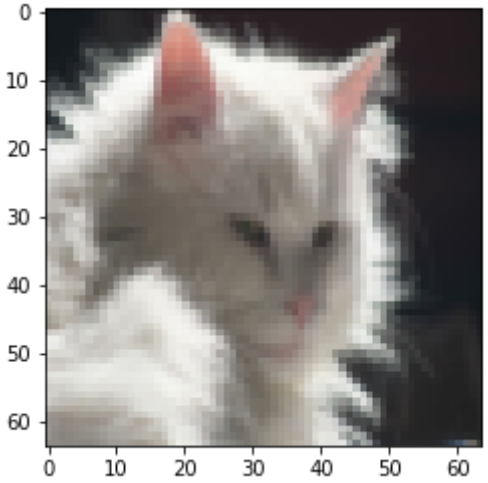
Comentários:

- Uma previsão errada de classe pode ser dectada pelos círculos vermelhos, pois quando a classe prevista é igual à classe real o círculo azul é colocado em cima do vermelho tapando-o.
- Provavelmente no grafico devem aparecer mais círculos azuis do que vermelhos indicando que existem mais acertos do que erros na previsão das classes.

Use o código a seguir, mudando a variável 'index', para você visualizar as imagens e a classe prevista dos exemplos de teste.

```
# Exemplo de classificação de uma imagem.
index = 8
plt.imshow(test_set_x[index,:].reshape((num_px, num_px, 3)))
print ("y = " + str(test_set_y[index]) + ", imagem é prevista como sendo uma '" + classes[np.squeeze(yy_prev[index])].decode("utf-8")
```

```
y = [1], imagem é prevista como sendo uma 'cat
```



Interpretação dos resultados:

O custo decrescendo durante o treinamento mostra que os parâmetros estão sendo ajustados de forma a que a rede está aprendendo os dados de treinamento. Como o custo obtido não é muito baixo, então, provavelmente é possível treinar ainda mais a rede para obter melhores resultados nos dados de treinamento.

Tente refazer as etapas 4.2, 4.3 e 4.4, aumentando o número de épocas para algo em torno de 5000, para retreinar a rede. Você verá que o custo de treinamento diminui até praticamente zero e a exatidão aumenta para quase 100%.

Importante: só tente treinar com mais épocas após entregar o seu trabalho, pois os resultados esperados são para o treinamento com 1000 épocas.

Contudo, nem sempre é bom treinar a rede até se obter custo perto de zero e exatidão 100% para os dados de treinamento. Em geral quando isso acontece o custo e a exatidão para os dados de teste pioraram. Nesse caso, pode ocorrer que a rede está memorizando os dados de treinamento e não generalizando a solução do problema. Veremos daqui algumas aulas como resolver esse problema. Mas podemos verificar esse fato retreinando a rede para um número maior de épocas.

5 - Desenvolvimento e teste da rede neural deep-learning

Nessa etapa do trabalho você vai configurar, treinar e testar uma rede neural deep learning.

Qualquer dúvida que você possa ter de como fazer essa parte do trabalho, ela pode ser sanada olhando as notas da Aula 7 - Ferramentas de Desenvolvmento.

Exercício #10:

Usando o Keras configure e crie uma rede neural com as seguintes caracteríticas:

- três camadas intermediárias com função de ativação tipo ReLu;
- número de neurônios das camadas intermediárias: 64, 32, 16;
- camada de saída deve possuir um único neurônio e ter função de ativação sigmóide.

Essa rede deve ser criada dentro de uma função (build_model) e os argumentos dessa função são: a dimensão dos dados de entrada e os números de neurônios das diversas camadas.

```
# PARA VOCÊ FAZER: configuração da rede deep-learning

# Importar do Keras modelos e camadas
from tensorflow.keras import models
from tensorflow.keras import layers

def build_model(data_shape,n1,n2,n3,n4):
    """
    Essa função configura uma rede neural deep-learnig
```

```
        Argumentos:
        data_shape = tuple com dimensões dos dados de entrada da rede
        n1 = número de neurônios da primeira camada
        n2 = número de neurônios da segunda camada
        n3 = número de neurônios da terceira camada
        n4 = número de neurônios da camada de saída

        Retorna: modelo da rede neural
        """

        np.random.seed(3)

        model = models.Sequential()

        # Adicione as camadas em seu modelo de RNA
        #### COMECE AQUI ### (≈ 4 linhas)
        model.add(Dense(units=n1, activation='relu', input_dim=data_shape[0]))
        model.add(Dense(units=n2, activation='relu'))
        model.add(Dense(units=n3, activation='relu'))
        model.add(Dense(units=n4, activation='sigmoid'))
        ### TERMINE AQUI ###

        return model

    # Redefine semente para geração de números aleatórios
    np.random.seed(3)

    # Dimensão dos dados de entrada
    data_shape = (12288,)

    # Definição dos números de neurônios das camadas
    #### COMECE AQUI ### (≈ 4 linhas)
    n1 = 64
    n2 = 32
    n3 = 16
    n4 = 1
    ### TERMINE AQUI ###
```

```
# Cria rede neural deep learning
#### COMECE AQUI ### (≈ 2 linhas)
rnadl = build_model(data_shape,n1,n2,n3,n4)
rnadl.summary()
### TERMINE AQUI ###
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 64)                786496    
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528       
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17        
=================================================================
Total params: 789,121
Trainable params: 789,121
Non-trainable params: 0
_________________________________________________________________
```

Saída esperada:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 64)                786496    
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_4 (Dense)              (None, 16)                528       
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17
```

```
=================================================================
Total params: 789,121
Trainable params: 789,121
Non-trainable params: 0
_________________________________________________________________
```

Exercício #11:

Repita os itens 4.2, 4.3 e 4.4 para a rede neural deep-learning. Use `rnadl` como nome da rede e acrescentando as letras `dl` nas variáveis de custo, exatidao e epocas.

Na célula abaixo compile e treine a sua rede deep-learning para 10 épocas. Esse treinamento é só para verificar se o programa está correto.

```
# PARA VOCÊ FAZER: compilação e treinamento da rede deep-learning

# Compilação da rede
### COMECE AQUI ### (≈ 2 linhas)
rnadl.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['acc'])
### TERMINE AQUI ###

# Teste de treinamento da rede
### COMECE AQUI ### (≈ 1 linha)
rnadl.fit(train_set_x, train_set_y, epochs=10, batch_size=209)
### TERMINE AQUI ###
```

```
Epoch 1/10
1/1 [==============================] - 0s 2ms/step - loss: 0.7014 - acc: 0.4402
Epoch 2/10
1/1 [==============================] - 0s 1ms/step - loss: 0.6595 - acc: 0.6411
Epoch 3/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6508 - acc: 0.6459
Epoch 4/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6431 - acc: 0.6459
Epoch 5/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6365 - acc: 0.6459
Epoch 6/10
1/1 [==============================] - 0s 4ms/step - loss: 0.6313 - acc: 0.6459
Epoch 7/10
1/1 [==============================] - 0s 3ms/step - loss: 0.6267 - acc: 0.6555
Epoch 8/10
1/1 [==============================] - 0s 1ms/step - loss: 0.6227 - acc: 0.6603
Epoch 9/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6196 - acc: 0.6603
Epoch 10/10
1/1 [==============================] - 0s 2ms/step - loss: 0.6157 - acc: 0.6603
<tensorflow.python.keras.callbacks.History at 0x7f2f285d73c8>
```

Saída esparada:

Epoch 1/10
209/209 [==============================] - 0s 1ms/sample - loss: 0.7339 - acc: 0.3206
Epoch 2/10
209/209 [==============================] - 0s 175us/sample - loss: 0.6939 - acc: 0.5072
Epoch 3/10
209/209 [==============================] - 0s 172us/sample - loss: 0.6786 - acc: 0.5933
Epoch 4/10
209/209 [==============================] - 0s 191us/sample - loss: 0.6661 - acc: 0.6268
Epoch 5/10
209/209 [==============================] - 0s 177us/sample - loss: 0.6588 - acc: 0.6316
Epoch 6/10

```
209/209 [==============================] - 0s 167us/sample - loss: 0.6529 - acc: 0.6459
Epoch 7/10
209/209 [==============================] - 0s 162us/sample - loss: 0.6479 - acc: 0.6507
Epoch 8/10
209/209 [==============================] - 0s 172us/sample - loss: 0.6433 - acc: 0.6507
Epoch 9/10
209/209 [==============================] - 0s 167us/sample - loss: 0.6391 - acc: 0.6555
Epoch 10/10
209/209 [==============================] - 0s 167us/sample - loss: 0.6359 - acc: 0.6555
```

Na célula abaixo treine a sua rede deep-learning por 1000 épocas. Use verbose = 0 e batch_size = 209.

```
# PARA VOCÊ FAZER: treinamento da rede deep-learning por 1000 épocas

### COMECE AQUI ### (≈ 1 linha)
historydl = rnadl.fit(train_set_x, train_set_y, epochs=1000, batch_size=209, verbose=0)
### TERMINE AQUI ###

# Vamos verificar quais variáveis foram salvas no processo de treinamento
historydl_dict = historydl.history
historydl_dict.keys()
```

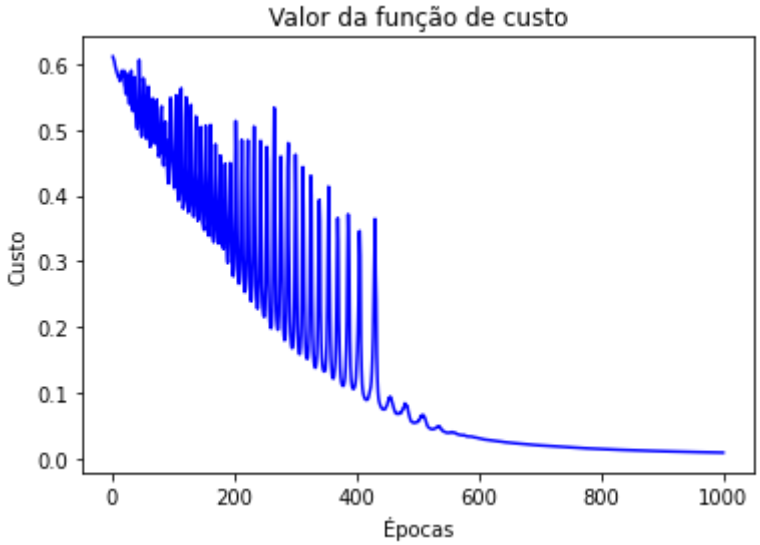
```
dict_keys(['loss', 'acc'])
```

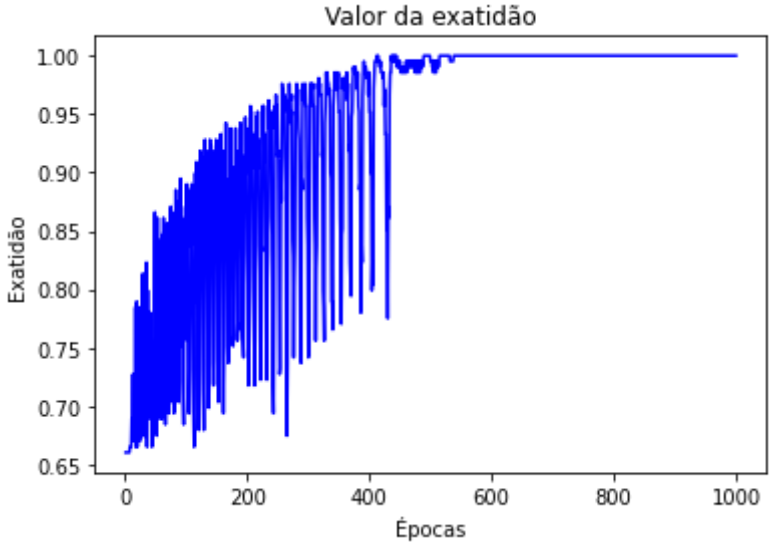
Exercício #12:

Na célula abaixo introduza os comandos para visualizar os resultados da sua rede deep-learning.

```
# PARA VOCÊ FAZER: visualização do resultado do treinamento da rede deep-learning

# Salva custo e exatidão em vetores
### COMECE AQUI ### (≈ 2 linhas)
```

```
    ### COMECE AQUI ### (≈ 2 linhas)
    custodl = historydl_dict['loss']
    exatidaodl = historydl_dict['acc']
    ### TERMINE AQUI ###

    # Cria vetor de épocas
    ### COMECE AQUI ### (≈ 1 linha)
    epocasdl = range(1, len(custo) + 1)
    ### TERMINE AQUI ###

    # Gráfico do custo em funçaõ das épocas
    ### COMECE AQUI ### (≈ 5 linhas)
    plt.plot(epocasdl, custodl, 'b')
    plt.title('Valor da função de custo')
    plt.xlabel('Épocas')
    plt.ylabel('Custo')
    plt.show()
    ### TERMINE AQUI ###

    # Gráfico da exatidão em função das épocas
    ### COMECE AQUI ### (≈ 5 linhas)
    plt.plot(epocasdl, exatidaodl, 'b')
    plt.title('Valor da exatidão')
    plt.xlabel('Épocas')
    plt.ylabel('Exatidão')
    plt.show()
    ### TERMINE AQUI ###
```





Exercício #13:

Na célula abaixo introduza os comandos para avaliar a sua rede deep-learning para os dados de treinamento e de teste.

```
# PARA VOCÊ FAZER: calculo do custo e exatidão para os dados de treinamento e de teste para a rede deep learning
```

```
# PARA VOCÊ FAZER: calculo do custo e exatidao para os dados de treinamento e de teste para a rede deep-learning

# Usando método evaluate calcule o custo e a exatidão dos dados de treinamento e depois apresente os resultados
### COMECE AQUI ### (≈ 2 linhas)
custo_e_metricasdl_train = rnadl.evaluate(train_set_x, train_set_y, batch_size=1)
print(custo_e_metricasdl_train)
### TERMINE AQUI ###

# Usando método evaluate calcule o custo e a exatidão dos dados de teste e depois apresente os resultados
### COMECE AQUI ### (≈ 2 linhas)
custo_e_metricasdl_test = rnadl.evaluate(test_set_x, test_set_y, batch_size=1)
print(custo_e_metricasdl_test)
### TERMINE AQUI ###
```

```
209/209 [==============================] - 0s 2ms/step - loss: 0.0084 - acc: 1.0000
[0.008403182961046696, 1.0]
50/50 [==============================] - 0s 2ms/step - loss: 1.0731 - acc: 0.6800
[1.0731061697006226, 0.6800000071525574]
```

Saída esperada:

209/209 [==============================] - 0s 482us/sample - loss: 0.0115 - acc: 1.0000
[0.011545336649320913, 1.0]
50/50 [==============================] - 0s 399us/sample - loss: 1.0013 - acc: 0.7400
[1.0013008093833924, 0.74]

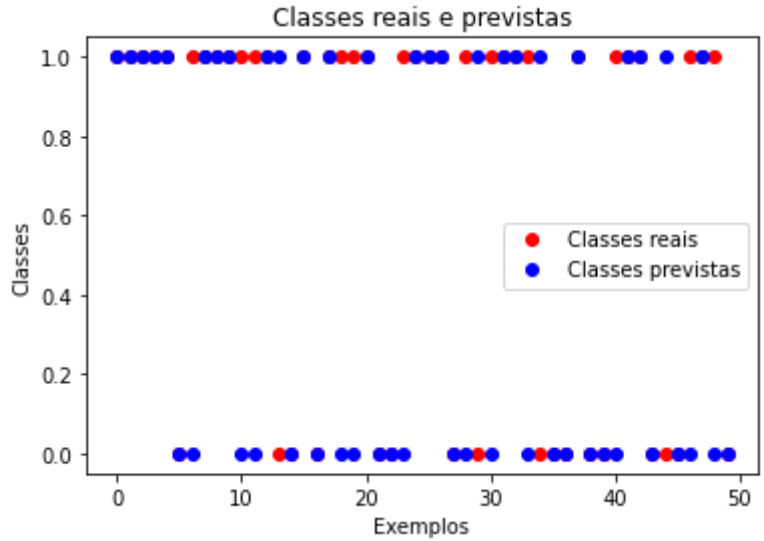
Exercício #14:

Na célula abaixo introduza os comandos para clcular as classes previstas para os dados de teste.

```
# PARA VOCÊ FAZER: calculo das classes previstas dos dados de teste para a rede deep-learning

# Usando método predict calcule as classes previstas
### COMECE AQUI ### (≈ 2 linhas)
y_prevdl = rnadl.predict(test_set_x, batch_size=1)
yy_prevdl = np.round(y_prevdl)
```

```
### TERMINE AQUI ###

# Transforma saida prevista em números inteiros
yy_prevdl = yy_prevdl.astype(int)

# Gráfico das classes reais e previstas
### COMECE AQUI ### (≈ 7 linhas)
plt.plot(test_set_y, 'ro', label='Classes reais')
plt.plot(yy_prev, 'bo', label='Classes previstas')
plt.title('Classes reais e previstas')
plt.xlabel('Exemplos')
plt.ylabel('Classes')
plt.legend()
plt.show
### TERMINE AQUI ###
```

```
<function matplotlib.pyplot.show>
```



Saída esperada:

O número de erros de classificação é igual a 12 (ou 13), representado por 12 (13) bolinhas vermelhas, o que significa 76% (74%) dos 50 exemplos de teste.

Interpretação dos resultados:

Se você fez tudo correto, então, os resultados obtidos do custo e da exatidão para os dados de teste são melhores para a rede neural deep-learning do que para a rede neural rasa, como era de se esperar.

Observe que a rede deep learning possui quase o mesmo número de parâmetros do que a rede rasa, mas obtém resultados mais satisfatórios.

O que fez com que os resultados da rede deep learning fossem muito melhores do que a rede rasa? O número de camadas, o tipo de função de ativação, ou simplesmenete o número de parâmetros?

O que é importante lembrar:

- A escolha do número de neurônios das camadas da rede é muito importane
- O tipo de função de ativação usada pode fazer diferença nos resultados
- Não é uma tarefa fácil desenvolver uma rede neural que apresenta um desempenho bom, mas não é impossível

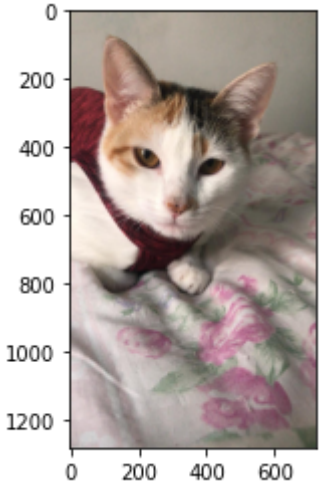
6 - Teste as redes que você desenvolveu com sua própria imagem

Você pode usar qualquer imagem e verificar se as suas redes neurais são eficientes. Para fazer isso deve fazer o seguinte: 1. Fazer upload do arquivo da imagem no seu Colab; 2. Colocar o nome do arquivo na variável `my_image`; 3. Executar a célula de código abaixo e verificar se a rede acerta (1 = imagem mostra gato, 0 = imagem não tem gato)!

```
# PARA VOCÊ FAZER: teste da rede com suas imagens

## COMECE AQUI ## (coloque o nome do arquivo com a sua imagem)
my_image = 'gato1.jpeg'
## TERMINE AQUI ##

# Pré-processamento da imagem para acertar dimensões.
fname = my_image
image = Image.open(fname)
my_image = image.resize((num_px, num_px), Image.ANTIALIAS)
my_image = np.array(my_image)
```

```
my_image = my_image.reshape((1, num_px*num_px*3))
my_image = my_image.astype(float)

# Previsão da rede neural
y_prev_myimage = rnadl.predict(my_image)
my_predicted_image = np.round(y_prev_myimage)
my_predicted_image = my_predicted_image.astype(int)

plt.imshow(image)
print("y = " + str(np.squeeze(my_predicted_image)) + ", sua rede prevê \"" + classes[int(np.squeeze(my_predicted_image)),].decode("ut
```

```
y = 1, sua rede prevê "cat" picture.
```



Finalmente, como sugestão, tente fazer alterações de parâmetros e executar o notebook novamente. Mas lembre-se de salvar o seu trabalho em um arquivo pdf e enviar para avaliação antes de modificar o notebook e realizar os seus testes.

Algumas sugestões do que alterar:

- Número de camadas;
- Número de neurônios nas camadas;
- Funções de ativação;
- Tente impor um taxa de aprendizado;

- Tente outras formas de normalizar os dados.