RedeNeuralConvolucional

August 6, 2019

1 Rede Neural Convolucional para Reconhecimento de Imagens de Cães e Gatos

Nesse documento mostraremos como construir uma rede neural convolucional para reconhecimento de imagens de cães e gatos, para isso utilizaremos a biblioteca Keras com o TensorFlow como backend para facilitar a nossa implementação. Essas são bibliotecas muito populares na área de inteligência artificial e disponibilizam diversas ferramentas que agilizam no desenvolvimento do projeto.

1.1 O que é uma Rede Neural Convolucional?

Uma rede neural convolucional (CNN do inglês Convolutional Neural network ou ConvNet) é uma classe de rede neural artificial do tipo feed-forward, que pode captar uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos/objetos da imagem e ser capaz de diferenciar um do outro; vem sendo aplicada com sucesso no processamento e análise de imagens digitais.

O pré-processamento exigido em uma ConvNet é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os filtros são feitos à mão, com treinamento suficiente, as ConvNets têm a capacidade de aprender esses filtros/características. Ela usa uma variação de perceptrons multicamada desenvolvidos de modo a demandar o mínimo pré-processamento possível. A arquitetura de uma ConvNet é análoga àquela do padrão de conectividade de neurônios no cérebro humano e foi inspirada na organização do Visual Córtex.

Os neurônios individuais respondem a estímulos apenas em uma região restrita do campo visual conhecida como Campo Receptivo. Uma coleção desses campos se sobrepõe para cobrir toda a área visual. Uma ConvNet é capaz de capturar com sucesso as dependências espaciais e temporais em uma imagem através da aplicação de filtros relevantes. A arquitetura executa um melhor ajuste ao conjunto de dados da imagem devido à redução no número de parâmetros envolvidos e à capacidade de reutilização dos pesos. Em outras palavras, a rede pode ser treinada para entender melhor a sofisticação da imagem.

1.2 Metodologia

Nós utilizaremos nesse projeto as ferramentas que o Anaconda nos oferece, juntamente com a IDE Spyder. Primeiramente algumas bibliotecas precisam ser instaladas:

- keras
- tensorFlow
- numpy

1.2.1 Construção da rede neural

Após instaladas, começaremos nosso código importando algumas ferramentas dessas bibliotecas:

```
In [6]: from keras.models import Sequential
        from keras.layers import Conv2D
        from keras.layers import MaxPooling2D
        from keras.layers import Flatten
        from keras.layers import Dense
```

Importado nossas bibliotecas, inicializaremos nossa rede:

```
In [59]: rede_neural = Sequential()
```

Agora adicionaremos a primeira camada convolucional da nossa rede e definiremos os parâmetros para o formato dos dados de entrada e a função de ativação. Usaremos 64 features para um array 2D e definiremos o nosso array com o formato de 3x3.

Todas a imagens de entrada de 64x64 pixels serão convertidas em um array 3D, pois as imagens coloridas RGB possuem 3 canais de cores.

```
In [60]: rede_neural.add(Conv2D(64, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
```

Logo depois da primeira camada, adicionaremos um agrupamento (pooling) para reduzir o tamanho do mapa de features obtida e descartar informações que não são tão relevantes.

Nesse caso utilizaremos o MaPooling que reduzirá pela metade o mapa e deixará somente o pontos característicos mais relevantes.

```
In [9]: rede_neural.add(MaxPooling2D(pool_size = (2, 2)))
```

Feito isso, adicionaremos uma segunda camada convolucional para tornar nossa rede mais profunda juntamente com outra de pooling. Porém dessa vez adicionaremos somente 32 features na camada convolucional.

```
In [10]: rede_neural.add(Conv2D(32, (3, 3), activation = 'relu'))
         rede_neural.add(MaxPooling2D(pool_size = (2, 2)))
```

Ao fim de nossas camadas convolucionais, aplicaremos um Flatten para converter os dados que estão em uma estrutura 2D para uma estrutura 1D.

```
In [11]: rede_neural.add(Flatten())
```

Nosso próximo passo é conectar todas as camadas. Utilizaremos a função de ativação ReLu em uma camada e a função de ativação Sigmóide na camada de saída para obter as probabilidades da imagem ser de gato ou de cachorro.

Dificilmente um modelo terá 100% de acertos, pois ele apenas calcula uma probabilidade.

```
In [12]: rede_neural.add(Dense(units = 128, activation = 'relu'))
         rede_neural.add(Dense(units = 1, activation = 'sigmoid'))
```

Finalmente após construirmos todas as camadas da nossa rede neural, iremos compilar ela. Utilizaremos o otimizador "Adam" e uma função loss "entropia binária cruzada" como parâmetros para esse processo. E por fim a nossa métrica será a acurácia, pois ela é nossa principal preocupação.

```
In [14]: rede_neural.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accur
```

1.2.2 Treinamento

Pré-Processamento Antes de começarmos a treinar nossa rede, precisamos primeiro pré-processar nossas imagens. Essa etapa é uma etapa crucial para o funcionamento do nosso projeto pois ele serve para normalizar os dados que servirão de entrada para a nossa rede.

Para isso utilizaremos a função ImageDataGenerator() da biblioteca Keras para ajustar alguns parâmetros das imagens de treino e de validação.

```
In [15]: from keras.preprocessing.image import ImageDataGenerator

         train_datagen = ImageDataGenerator(rescale = 1./255,
                                            shear_range = 0.2,
                                            zoom_range = 0.2,
                                            horizontal_flip = True)

         test_datagen = ImageDataGenerator(rescale = 1./255)
```

Agora aplicaremos os objetos criados anteriormente para pré-processar nossas imagens de treino e de validação.

```
In [16]: training_set = train_datagen.flow_from_directory('dogs-and-cats-dataset/training_set',
                                                         target_size = (64, 64),
                                                         batch_size = 32,
                                                         class_mode = 'binary')

         test_set = test_datagen.flow_from_directory('dogs-and-cats-dataset/test_set',
                                                     target_size = (64, 64),
                                                     batch_size = 32,
                                                     class_mode = 'binary')
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

1.2.3 Treinamento da rede

Para treinar nossa rede, usamos 8000 passos em nosso conjunto de treinamento para cada época, e escolhemos 2000 etapas de validação para as imagens de validação. A quantidade de épocas foi definida para o valor de 7.

Esse processo de treinamento pode ser um pouco demorado dependendo da capacidade de cada computador.

Para um processo mais rápido ou para computadores com baixa capacidade de processamento, o valor de épocas pode ser reduzido para 5, mas isso afetará um pouco a acurácia da sua rede. Diminuir os dados de treinamento também pode ser uma opção.

```
In [ ]: rede_neural.fit_generator(training_set,
                                  steps_per_epoch = 8000,
                                  epochs = 7,
                                  validation_data = test_set,
                                  validation_steps = 2000)
```

Depois de ter treinado sua rede iremos salvá-lo em um arquivo .json. Seus pesos serão salvos em um arquivo .h5 para serem utilizados depois sem ser necessário treiná-lo de novo.

```
In [20]: rede_neural_json = rede_neural.to_json()
         with open("ARQUIVO.json","w") as json_file:
             json_file.write(rede_neural_json)

         rede_neural.save_weights("ARQUIVO.h5")
```

1.3 Resultados

Agora iremos testar o nosso modelo treinado, fazendo-o predizer imagens que ele nuca viu antes.

Primeiro carregaremos o nosso modelo e seus pesos salvos.

```
In [29]: from keras.models import model_from_json
         #Informe o diretório em que ser arquivo foi salvo
         json_file = open("ModeloConv2D_7e.json","r")
         modelo_json = json_file.read()
         json_file.close()

         modelo = model_from_json(modelo_json)
         modelo.load_weights("ModeloConv2D_7e.h5")
         modelo.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy']
```

Vamos criar agora uma função para carregar a nossa imagem com as mesmas dimensões usadas nas imagens de treino e depois convertê-la para um array. Depois disso faremos nosso modelo prever se a imagem contém um gato ou um cachorro.

```
In [30]: import numpy as np
         from IPython.display import Image
         from keras.preprocessing import image
         #path é o camiho onde sua imagem está
         def read_image(path):
             test_image = image.load_img(path, target_size = (64,64))
             test_image = image.img_to_array(test_image)
             test_image = np.expand_dims(test_image, axis = 0)
             return test_image

In [40]: test_image = read_image("teste/1.jpg")

         result = modelo.predict(test_image)

         if result[0][0] == 1:
             prediction = 'Cachorro'
         else:
             prediction = 'Gato'

         print("Imagem reconhecida como: ", prediction)
         Image(filename='teste/1.jpg')
```

```
Imagem reconhecida como:  Gato
```

Out[40]:

Nossa primeira imagem de teste foi reconhecida com sucesso. Isso se deve ao fato de que nossa rede conseguiu aprender com base no que ele viu antes na fase de treinamento, padrões que consegue identificar se a imagem é de um gato ou cachorro.

Esse padrões são nada menos que pesos que representam uma determinada característica da imagem, como as patas ou as orelhas por exemplo.

In [41]:
```
test_image = read_image("teste/2.jpg")

result = modelo.predict(test_image)

if result[0][0] == 1:
    prediction = 'Cachorro'
else:
    prediction = 'Gato'

print("Imagem reconhecida como: ", prediction)
Image(filename='teste/2.jpg')
```

```
Imagem reconhecida como:  Cachorro
```

Out[41]:

A segunda imagem também conseguiu ser reconhecida com sucesso.

In [43]:
```
test_image = read_image("teste/3.jpg")

result = modelo.predict(test_image)

if result[0][0] == 1:
    prediction = 'Cachorro'
else:
    prediction = 'Gato'

print("Imagem reconhecida como: ", prediction)
Image(filename='teste/3.jpg')
```

```
Imagem reconhecida como:  Gato
```

Out[43]:

Nossa terceira imagem, mesmo com o gato estar usando um chapéu, a nossa rede neural conseguiu identificá-lo corretamente.

```
In [44]: test_image = read_image("teste/4.jpg")

         result = modelo.predict(test_image)

         if result[0][0] == 1:
             prediction = 'Cachorro'
         else:
             prediction = 'Gato'

         print("Imagem reconhecida como: ", prediction)
         Image(filename='teste/4.jpg')
```

```
Imagem reconhecida como:  Cachorro
```

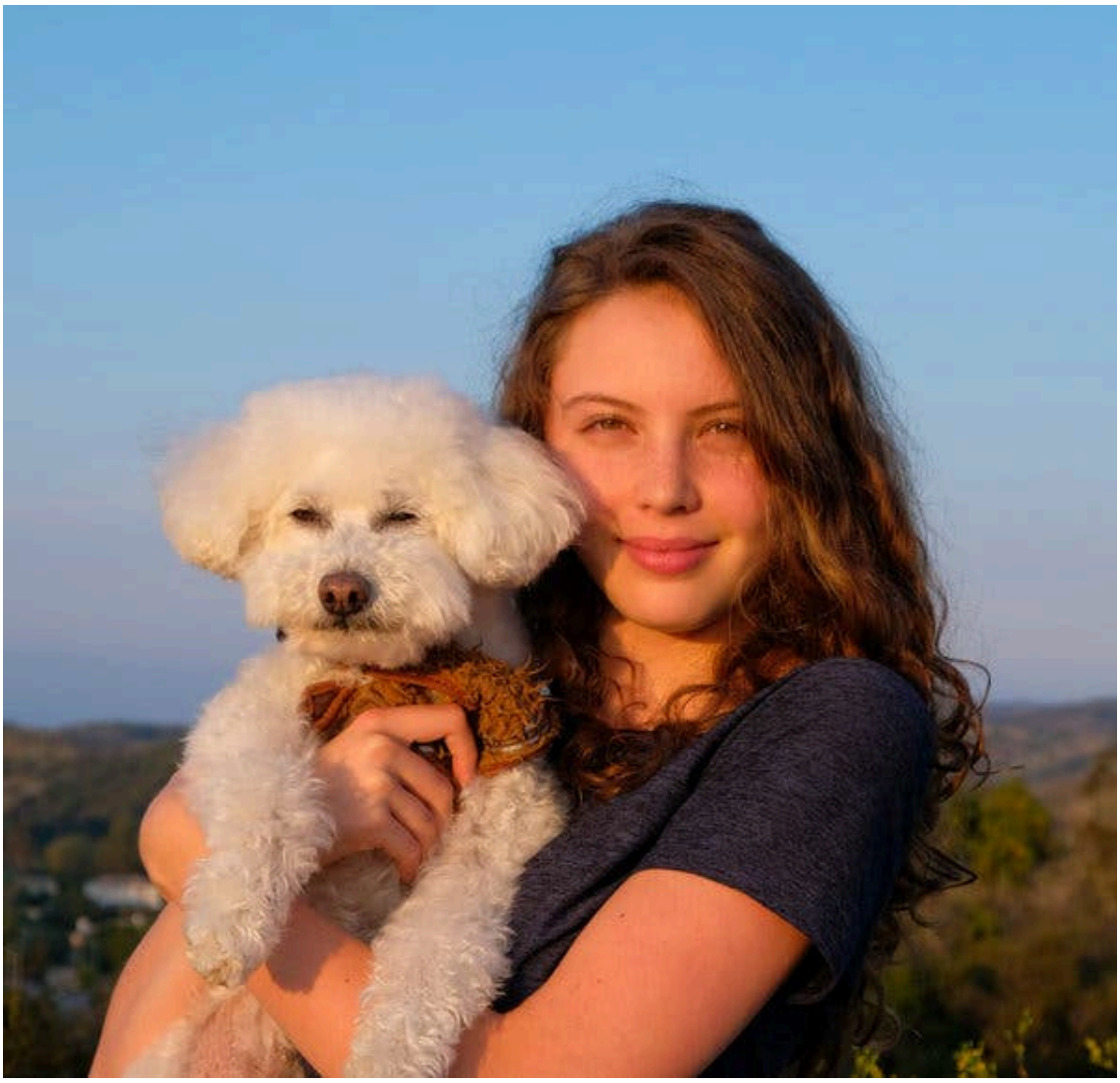
```
Out[44]:
```

A quarta imagem também foi reconhecida com sucesso.

```
In [46]: test_image = read_image("teste/5.jpg")

         result = modelo.predict(test_image)

         if result[0][0] == 1:
             prediction = 'Cachorro'
         else:
             prediction = 'Gato'

         print("Imagem reconhecida como: ", prediction)
         Image(filename='teste/5.jpg')
```

```
Imagem reconhecida como:  Cachorro
```

```
Out[46]:
```



Nossa quinta imagem possui um cachorro ao lado de uma mulher, mesmo assim a nossa rede neural conseguiu identificá-lo corretamente.

```
In [47]: test_image = read_image("teste/6.jpg")

         result = modelo.predict(test_image)

         if result[0][0] == 1:
             prediction = 'Cachorro'
         else:
             prediction = 'Gato'

         print("Imagem reconhecida como: ", prediction)
         Image(filename='teste/6.jpg')
```

```
Imagem reconhecida como:  Cachorro
```

Out[47]:

Na sexta imagem a nossa rede neural não conseguiu identificá-lo corretamente, isso pode se dar por diversos fatores como o número de dados para treino não ter sido o suficiente, ou até mesmo a topologia escolhida para nossa rede neural não ter sido a mais adequada para esse problema. Porém isso não é algo para se desesperar, pois como foi falado anteriormente, dificilmente uma rede neural irá acertar tudo.

In [61]:
```
test_image = read_image("teste/7.jpg")

result = modelo.predict(test_image)

if result[0][0] == 1:
    prediction = 'Cachorro'
else:
    prediction = 'Gato'

print("Imagem reconhecida como: ", prediction)
Image(filename='teste/7.jpg')
```

```
Imagem reconhecida como:  Cachorro
```

Out[61]:

E para finalizar, nossa sétima imagem também não foi reconhecida corretamente.

1.4 Conclusão

O resultado final do nosso projeto foi de uma acurácia de aproximadamente 98% em nosso conjunto de treino e de 87% em nosso conjunto de testes.

Após isso, das nossas 7 imagens testadas, apenas 5 foram reconhecidas corretamente e 2 não.

Algumas sugestões para serem testadas a fim de obter um resultado melhor, são:

- Aumentar a resolução das imagens, isso faz com que a nossa rede consiga reconhecer características mais precisamente.
- Aumentar o número de épocas para tornar o aprendizado mais profundo.

- Alterar a topologia da nossa rede adicionando mais uma camada convolucional, ou adicionando o número de neurônios.
- Alterar alguns parâmetros da rede.