Criando uma Rede Neural Convolucional com TensorFlow e Google Colab

Izadora Dourado Berti idberti@inf.ufrgs.br
Pedro Wachsmann Schanzer de Oliveira pedroschanzer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul Porto Alegre, Rio Grande do Sul, Brasil

Resumo

Esse documento aborda tópicos que foram discutidos na disciplina de Tópicos Especiais de Computação III - "State-of-the-Art in Computer Graphics Seminar" e apresenta uma perspectiva de democratização da tecnologia através do uso de ferramentas gratuitas para a criação e teste de uma Rede Neural Convolucional.

1. Introdução:

A classificação de imagens pode ser feita através de diversos modelos de análise. Um dos mais populares atualmente é a Rede Neural Convolucional (CNN), na qual diferentes camadas mapeiam recursos únicos na imagem e extraem as principais informações, classificando a imagem através desses recursos.

Com a programação tradicional não é possível construir soluções escaláveis para problemas como visão computacional, uma vez que não é viável escrever um algoritmo que seja generalizado o suficiente para identificar a natureza das imagens. Com o aprendizado de máquina, podemos construir uma aproximação que seja suficiente para casos de uso, treinando um modelo para determinados exemplos e fazendo previsões para dados não vistos.

Rede Neural Convolucional (CNN) é um tipo especial de rede neural profunda que tem um desempenho impressionante em problemas de visão computacional, como classificação de imagens, detecção de objetos, etc. Em um mundo cada vez mais conectado à nuvem de dados, é essencial que sejam promovidas ferramentas gratuitas e acessíveis para democratizar o uso dessa tecnologia e gerar os debates éticos necessários para que o seu uso seja qualificado e tenha como resultado impacto socioambiental positivo. Como a performance do treinamento de inteligência artificial depende da qualidade do sistema do usuário, existem plataformas que permitem a programação remota dessa tecnologia. Neste artigo, vamos demonstrar como é intuitiva, gratuita e acessível a utilização da plataforma Google Colab para criar um classificador de imagens utilizando Tensorflow para implementar uma CNN para classificar imagens de cães e gatos.

2. Descrições:

Tensorflow[1] é uma plataforma de código aberto para aprendizado de máquina feita pela Google AI. Através do uso dela e da ferramenta Google Colab[2], é possível implementar uma Rede Neural Convolucional e treiná-la, gerando testes de performance para diferentes conjuntos de dados e modelos de aprendizado. O acesso aos recursos da plataforma é gratuito e existem diversos tutoriais em diferentes línguas para guiar o ensino desta tecnologia. O processamento é feito através do (GPU) de back-end do Google Compute Engine em Python 3, deste modo com uma conexão a internet podemos programar em Python de forma remota e obter performances razoáveis.O primeiro passo para programar é importar a biblioteca Tensorflow e definir um conjunto de dados para treinamento. Com a configuração inicial do ambiente e dos dados, podemos pensar no modelo de Aprendizado de Máquina que será implementado.

O modelo de Rede Neural Convolucional (CNN) é baseado na operação de convolução, na qual a convolução entre dois sinais gera um terceiro que representa a soma do produto desses dois sinais iniciais. Para reconhecer recursos de imagens a rede neural convolucional é feita através da convolução entre um filtro de recursos e uma imagem do conjunto de dados, resultando em uma matriz reduzida com as informações sobre aquele determinado recurso. A soma de diversas camadas de recursos forma os níveis de profundidade dessa rede neural e a imagem é identificada através do mapa de recursos analisados, construído com várias camadas de convolução, camadas de pool e camadas densas.

Quando falamos em reconhecimento e classificação de imagens, as entradas são usualmente matrizes tridimensionais com altura, largura e profundidade, determinada pela quantidade de canais de cores. Em geral as imagens utilizam três canais, RGB, com os valores de cada pixel. Uma CNN pode ser dividida em duas partes: extração de características (*Conv*, *Padding, Relu*, *Pooling*) e uma rede neural tradicional.

A ideia da camada de convolução é transformar a imagem de entrada para extrair características (ex. Orelhas, nariz, pernas de gatos e cães) para distingui-los corretamente. Isso é feito convulsionando a imagem com um kernel (filtro ou detector de recurso). Um kernel é especializado para extrair certos recursos. É possível aplicar vários kernels a uma única imagem para capturar vários recursos.

As convoluções funcionam como filtros que enxergam pequenos quadrados e vão varrendo toda a imagem captando os traços mais marcantes. A profundidade da saída de uma convolução é igual a quantidade de filtros aplicados. Quanto mais profundas são as camadas das convoluções, mais detalhados são os traços identificados com o "activation map".

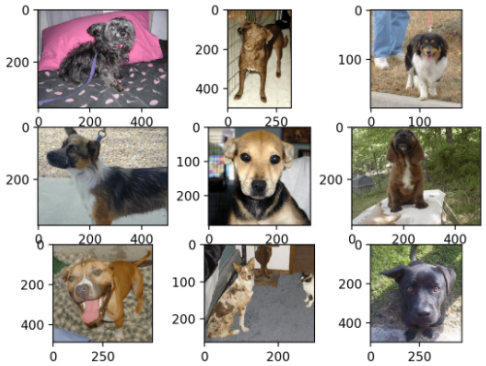
O filtro, que também é conhecido por kernel, é formado por pesos inicializados aleatoriamente, atualizando-os a cada nova entrada durante o processo de "backpropagation". A pequena região da entrada onde o filtro é aplicado é chamada de "receptive field".

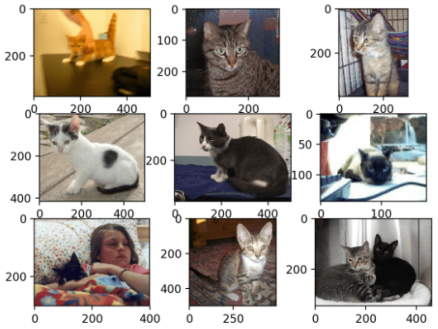
Criando uma Rede Neural Convolucional com TensorFlow e Google Colab

Izadora Dourado Berti idberti@inf.ufrgs.br
Pedro Wachsmann Schanzer de Oliveira pedroschanzer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul Porto Alegre, Rio Grande do Sul, Brasil

CNN é uma rede neural profunda que precisa de muito poder de computação para treinamento. Além disso, para obter precisão suficiente, deve haver um grande conjunto de dados para construir um modelo generalizado para dados invisíveis. Como citado anteriormente, para a implementação, fizemos uso do Colaboratory ou "Colab", que é uma plataforma da Google para fins de pesquisa, que permite escrever código Python no navegador, utilizando TensorFlow para desenvolvimento e treinamento de redes neurais, experimentos com TPUs, divulgação de pesquisas em IA e criação de tutoriais. Com o Colab, é possível importar um conjunto de dados de imagem, treinar um classificador de imagens dentro dele e avaliar o modelo, além de que suporta hardware habilitado para GPU, o que também dá um grande impulso para o treinamento. O conjunto de dados utilizados para esse modelo de aprendizado de máquina foi o 'cats_and_dogs.zip', com imagens de cães e gatos.

Abaixo estão imagens que servem de exemplo de imagens presentes no conjunto de dados.





3. Implementação

Utilizando a plataforma Google Colab, em uma mesma página é possível conectar a um ambiente de execução e escrever trechos de código, importando a biblioteca TensorFlow para classificar um conjunto de dados.
O código foi escrito em 6 trechos, descritos abaixo.

1º Trecho: Utlizando o comando !wget, baixamos para o ambiente o conjunto de dados compactados 'cats_and_dogs.zip'. No caso, esse conjunto de dados contém 2.000 imagens '.jpg' de cães e gatos.

```
!wget --no-check-certificate \
https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip\
-O /tmp/cats_and_dogs_filtered.zip
```

2º Trecho: Esse conjunto de dados compactados é extraído para o ambiente.

```
import os
import zipfile

local_zip = '/tmp/cats_and_dogs_filtered.zip'

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

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

Izadora Dourado Berti idberti@inf.ufrgs.br
Pedro Wachsmann Schanzer de Oliveira pedroschanzer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul Porto Alegre, Rio Grande do Sul, Brasil

3º Trecho: Instância o diretório base como 'cats_and_dogs'. instancia duas novas partições no diretório, uma de treinamento e outra de validação. Configura quais recursos estarão relacionados a cada um desses diretórios (gatos ou cachorros).

```
base_dir = '/tmp/cats_and_dogs_filtered'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

train_cats_dir =os.path.join(train_dir, 'cats')
train_dogs_dir =os.path.join(train_dir, 'dogs')

validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
```

4º Trecho, Parte 1: Configura como será o teste e a validação (quais parâmetros serão executados): ao carregar os dados, fazemos isso em 20 lotes de imagens (“batch_size”), com classe binária e todos eles são redimensionados para o tamanho 150x150. Se houver imagens em tamanhos diferentes, isso vai consertar. É extremamente importante tratar os dados antes de executar as operações matemáticas, para evitar que dados mal formatados possam gerar desvios e erros no treinamento da rede neural.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))
# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=20,
                                                         class_mode  = 'binary',
                                                         target_size = (150, 150))
```

4º Trecho, Parte 2: Configura o modelo de convolução a ser feito. A convolução é feita em camadas de filtros no formato de matrizes de diferentes tamanhos. 1º matriz 3x3 e “Filter Count” de 16 com “MaxPooling” de 2x2, 2º matriz 3x3 e “Filter Count” de 32 com “MaxPooling” de 2x2, 3º matriz 3x3 e “Filter Count” de 64 com “MaxPooling” de 2x2 e 4º matriz 3x3 e “Filter Count” 1024 com “MaxPooling” de 2x2. O filtro terá tamanho 3x3. O “Filter Count” é o contador de filtros, quanto mais filtros, mais preciso o modelo. O “MaxPooling” significa que será gerada uma matriz final 2x2 contendo apenas os valores máximos do filtro. A “activation” será feita por uma “Rectified Linear Unit” (ReLU), aplicando não linearidade para interpretar imagens complexas. A densidade de neurônios desta camada será de apenas um neurônio, aplicando a função sigmoid. Ao rodar esse trecho de código, obtemos a mensagem: "Found 2000 images belonging to 2 classes”. Isso confirma que o modelo fez a operação correta de processamento de imagem.

```
model = tf.keras.models.Sequential([
                                    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150,150,3)),
                                    tf.keras.layers.MaxPooling2D(2,2),
                                    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
                                    tf.keras.layers.MaxPooling2D(2,2),
                                    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
                                    tf.keras.layers.MaxPooling2D(2,2),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(1024, activation='relu'),
                                    tf.keras.layers.Dense(1, activation='sigmoid')


])
```

Izadora Dourado Berti idberti@inf.ufrgs.br
Pedro Wachsmann Schanzer de Oliveira pedroschanzer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul Porto Alegre, Rio Grande do Sul, Brasil

5º Trecho: Configura como será feita a otimização do treinamento. 'binary_crossentropy' para modelo de classificação, pois lida com duas classes binárias, 'RMSprop' para parâmetro de otimização e como métrica para melhoramento de desempenho "acc" (acurácia/precisão).

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics = ['acc'])
```

6º trecho: Configura como vamos carregar os dados no treinamento. Epochs simboliza o número de passagens completas pelo conjunto de dados e nesse treino foi configurado como 15. Após percorrer o conjunto de dados 15 vezes utilizando o modelo de rede neural convolucional desejado, obteremos os resultados do teste de performance.

```
history = model.fit_generator(train_generator,
                              validation_data=validation_generator,
                              steps_per_epoch=100,
                              epochs=15,
                              validation_steps=50,
                              verbose=1)
```

4. Resultados Obtidos

Após rodar o **6º Trecho** final, nos sistemas descritos abaixo, obtivemos os seguintes resultados após 15 epochs:

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` i
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/15
100/100 [==============================] - 69s 677ms/step - loss: 1.0182 - acc: 0.5316 - val_loss: 0.6692 - val_acc: 0.5880
Epoch 2/15
100/100 [==============================] - 67s 675ms/step - loss: 0.6623 - acc: 0.6309 - val_loss: 0.5962 - val_acc: 0.7050
Epoch 3/15
100/100 [==============================] - 67s 669ms/step - loss: 0.5502 - acc: 0.7369 - val_loss: 0.5708 - val_acc: 0.7240
Epoch 4/15
100/100 [==============================] - 70s 696ms/step - loss: 0.4688 - acc: 0.7602 - val_loss: 0.7223 - val_acc: 0.6490
Epoch 5/15
100/100 [==============================] - 67s 673ms/step - loss: 0.3728 - acc: 0.8371 - val_loss: 0.6725 - val_acc: 0.7050
Epoch 6/15
100/100 [==============================] - 67s 671ms/step - loss: 0.2569 - acc: 0.8991 - val_loss: 0.8341 - val_acc: 0.7090
Epoch 7/15
100/100 [==============================] - 67s 673ms/step - loss: 0.1656 - acc: 0.9305 - val_loss: 0.9430 - val_acc: 0.6950
Epoch 8/15
100/100 [==============================] - 67s 670ms/step - loss: 0.1219 - acc: 0.9536 - val_loss: 1.0435 - val_acc: 0.7020
Epoch 9/15
100/100 [==============================] - 67s 670ms/step - loss: 0.0902 - acc: 0.9731 - val_loss: 1.2810 - val_acc: 0.7130
Epoch 10/15
100/100 [==============================] - 67s 667ms/step - loss: 0.0476 - acc: 0.9844 - val_loss: 1.5748 - val_acc: 0.7330
Epoch 11/15
100/100 [==============================] - 67s 669ms/step - loss: 0.0593 - acc: 0.9841 - val_loss: 1.7345 - val_acc: 0.7170
Epoch 12/15
100/100 [==============================] - 67s 670ms/step - loss: 0.0436 - acc: 0.9884 - val_loss: 1.7315 - val_acc: 0.7220
Epoch 13/15
100/100 [==============================] - 69s 690ms/step - loss: 0.1012 - acc: 0.9950 - val_loss: 1.8468 - val_acc: 0.7300
Epoch 14/15
100/100 [==============================] - 68s 677ms/step - loss: 0.0151 - acc: 0.9979 - val_loss: 2.1062 - val_acc: 0.7270
Epoch 15/15
100/100 [==============================] - 67s 675ms/step - loss: 0.1105 - acc: 0.9915 - val_loss: 2.2648 - val_acc: 0.7120
```

Fig. 3. Desempenho no teste de CNN no Sistema 1 após 15 epochs, onde o modelo obteve 99,15% de acerto no conjunto de treinamento e 71,20% de acerto no conjunto de validação.

Izadora Dourado Berti idberti@inf.ufrgs.br
Pedro Wachsmann Schanzer de Oliveira pedroschanzer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul Porto Alegre, Rio Grande do Sul, Brasil

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/15
100/100 [==============================] - 69s 675ms/step - loss: 1.9504 - acc: 0.4947 - val_loss: 0.6732 - val_acc: 0.6010
Epoch 2/15
100/100 [==============================] - 67s 673ms/step - loss: 0.6644 - acc: 0.6170 - val_loss: 0.6237 - val_acc: 0.6630
Epoch 3/15
100/100 [==============================] - 67s 671ms/step - loss: 0.5945 - acc: 0.6951 - val_loss: 0.6126 - val_acc: 0.6780
Epoch 4/15
100/100 [==============================] - 67s 672ms/step - loss: 0.4969 - acc: 0.7476 - val_loss: 0.5853 - val_acc: 0.6920
Epoch 5/15
100/100 [==============================] - 67s 670ms/step - loss: 0.4367 - acc: 0.8044 - val_loss: 0.6686 - val_acc: 0.6800
Epoch 6/15
100/100 [==============================] - 67s 671ms/step - loss: 0.3143 - acc: 0.8698 - val_loss: 0.7345 - val_acc: 0.7140
Epoch 7/15
100/100 [==============================] - 68s 676ms/step - loss: 0.1829 - acc: 0.9247 - val_loss: 0.9351 - val_acc: 0.7260
Epoch 8/15
100/100 [==============================] - 68s 676ms/step - loss: 0.1250 - acc: 0.9525 - val_loss: 0.8593 - val_acc: 0.7230
Epoch 9/15
100/100 [==============================] - 68s 676ms/step - loss: 0.0710 - acc: 0.9748 - val_loss: 1.2483 - val_acc: 0.7320
Epoch 10/15
100/100 [==============================] - 68s 675ms/step - loss: 0.1294 - acc: 0.9726 - val_loss: 1.3754 - val_acc: 0.7010
Epoch 11/15
100/100 [==============================] - 67s 675ms/step - loss: 0.0577 - acc: 0.9891 - val_loss: 1.7086 - val_acc: 0.7400
Epoch 12/15
100/100 [==============================] - 67s 675ms/step - loss: 0.1103 - acc: 0.9735 - val_loss: 1.5183 - val_acc: 0.7200
Epoch 13/15
100/100 [==============================] - 67s 672ms/step - loss: 0.0349 - acc: 0.9893 - val_loss: 1.8295 - val_acc: 0.7210
Epoch 14/15
100/100 [==============================] - 67s 674ms/step - loss: 0.0349 - acc: 0.9902 - val_loss: 1.6069 - val_acc: 0.7280
Epoch 15/15
100/100 [==============================] - 67s 674ms/step - loss: 0.0325 - acc: 0.9901 - val_loss: 1.5330 - val_acc: 0.7250
```

Fig. 4. Desempenho no teste de CNN no Sistema 2 após 15 epochs, onde o modelo obteve 99,01% de acerto no conjunto de treinamento e 72,50% de acerto no conjunto de validação.

Assim, notamos que ambos alunos, mesmo usando computadores e conexões a internet completamente diferentes, conseguiram resultados muito semelhantes na execução remota dessa Rede Neural Convolucional. A conclusão de todas as Epochs foi alcançada em 16 minutos e 52 segundos.

5. Conclusões

Muitas décadas atrás a Inteligência Artificial era tema discutido apenas na esfera acadêmica de poucas universidades de nível internacional. Atualmente, a democratização da tecnologia, tornou possível que diversos estudantes ao redor do mundo testassem modelos avançados de computação, compartilhando recursos através da internet. Em um curto intervalo de tempo foi possível preparar o ambiente, configurar o modelo de aprendizado de máquina, implementar um treinamento e testar seu desempenho em um conjunto de dados. Ferramentas como Google Colab e TensorFlow permitem que qualquer pessoa com acesso a internet e conhecimento de programação consiga iniciar seus estudos e testes nos mais diversos modelos de aprendizado de máquina.

Comparando nosso desempenho com o desempenho de um modelo muito similar disponível em um tutorial do TensorFlow [3] de Rede Neural Convolucional que utiliza o conjunto de dados CIFAR, classificação por meio de “labels” e 10 epochs, podemos identificar um desempenho similar com 80,88% de acerto no conjunto de treinamento e 71,57% de acerto no conjunto de validação. Isso se deve ao fato do modelo do tutorial ter utilizado uma quantidade menor de “Filter Count” com valores menores do que os utilizados no nosso teste. Além disso, utilizaram menos Epochs para treinar o modelo de aprendizado de máquina. Com isso conseguimos definir a importância desses parâmetros para melhorar o desempenho de redes neurais convolucionais.

Izadora Dourado Berti idberti@inf.ufrgs.br
Pedro Wachsmann Schanzer de Oliveira pedroschanzer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul Porto Alegre, Rio Grande do Sul, Brasil

6. Alternativas para desenvolvimento futuro

Nosso modelo terá um desempenho muito bom no conjunto de treinamento binário (cães e gatos) e um desempenho ruim para os dados invisíveis. Para resolver o problema de "overfitting", podemos adicionar regularização para evitar a super complexação do modelo ou podemos adicionar mais dados ao conjunto de treinamento para tornar o modelo mais generalizado para dados invisíveis. Como temos um conjunto de dados muito pequeno (2.000 imagens) para treinamento, adicionar mais dados deve resolver o problema. Coletar mais dados para treinar um modelo é difícil no aprendizado de máquina, pois é necessário pré-processar os dados novamente. Mas ao trabalhar com imagens, especialmente na classificação de imagens, não há necessidade de coletar mais dados. Isso pode ser corrigido com a técnica chamada de aumento de imagem.

Por mais que existam abordagens que visem solucionar essas falhas, os modelos de inteligência artificial são péssimos quando se tratam de dados nunca antes analisados e fora dos padrões estabelecidos em treinamento. O modelo tentará encaixar esse dado em algum rótulo já existente e caso não consiga, não saberá como classificar esse dado. É importante que os conjuntos de dados passem pela operação de tratamento e sejam formatados da maneira correta. Existem diversas pesquisas que buscam melhorar a qualidade de imagens disponíveis, diminuindo ruídos existentes e tornando o conjunto menos sujeito a falhas de treinamento. Yehoshua Y. Zeev, formulou algoritmos para que imagens possam ser tratadas de maneira a não ocorrer perda de informação [4]. Deste modo, cada imagem identificada pode ser melhorada, garantindo a coesão do banco de imagens que treinará a rede neural convolucional. Além disso, foram desenvolvidos algoritmos de aprendizado não-supervisionado para clusterização das imagens com ruídos semelhantes, gerando uma "digital de ruído" para cada imagem e diminuindo as chances de que ruídos possam impactar negativamente no processamento dos mapas de recursos e na precisão da rede neural.

Modelos populares de conjunto de dados estão cheios de viés e ferramentas utilizadas por milhões de clientes no mundo têm essas falhas éticas ocorrendo de maneira exponencial, pois os dados desses usuários muitas vezes são armazenados e analisados por modelos sujeitos a falhas. A democratização do acesso a essa tecnologia tende a aumentar o número de conjuntos de dados disponíveis para pesquisa e além disso, torná-los mais diversos, com dados provenientes de diferentes sociedades e culturas.

7. Referências Bibliográficas:

[1] Plataforma TensorFlow (https://www.tensorflow.org/)
[2] Plataforma Google Colab (https://colab.research.google.com/notebooks/intro.ipynb)
[3] Tutorial TensorFlow de CNN (https://www.tensorflow.org/tutorials/images/cnn)
[4]Forward-and-Backward Diffusion Processes for Adaptive Image Enhancement and Denoising Guy Gilboa, Nir Sochen, and Yehoshua Y. Zeevi (https://bit.ly/3kbGejU)