CNN desenvolvida com Tensorflow/CIFAR10

Profa. Dra. Roseli Aparecida Francelin Romero

Disciplina: SCC 0270 - Redes Neurais

Lucas Yudi Sugi - 9293251

Introdução

As redes neurais convolucionais são atualmente o estado da arte para as tarefas relacionadas a classificação de imagens. Com elas está sendo possível realizar predições de imagens com uma grande acurácia, fato que antigamente era impensável pois a complexidade do problema era grande demais e as técnicas muito difíceis de serem implementadas.

Objetivos

O objetivo deste documento é ilustrar a confecção de uma CNN utilizando a API do Google chamada TensorFlow, assim como verificar o poder de generalização da mesma em imagens que não fazem parte do dataset cifar10. O código produzido é baseado neste tutorial:

https://www.tensorflow.org/tutorials/estimators/cnn

Base dados utilizada:

https://www.cs.toronto.edu/~kriz/cifar.html

Arquitetura da rede

A rede neural convolucional possui a seguinte arquitetura:

- Convolutional Layer 1:
  - Filters: 64
  - Kernel size: [3,3]
  - Activation function: Relu
- Pooling layer 1:
  - Stride = 2
  - Pool size: [2,2]
- Convolutional Layer 2:
  - Filters: 128
  - Kernel size: [3,3]
  - Activation function: Relu
- Pooling layer 2:
  - Stride = 2
  - Pool size: [2,2]
- Convolutional Layer 3:
  - Filters: 256

    - Kernel size: [3,3]
    - Activation function: Relu
- Pooling layer 3:
    - Stride = 2
    - Pool size: [2,2]
- Flatten:
    - Vetor de dimensão [bath_size,4096] (Redimensionamento necessário para as camadas densas)
- Dense layer 1:
    - Units: 512
    - Activation function: Relu
- Dense layer 2:
    - Units: 1024
    - Activation function: Relu
- Dropout:
    - 20% de chance do nó ser desligado na segunda camada densa.
- Dense layer 3:
    - Units: 10
    - Activation function: Linear

Note que a arquitetura implementada é uma versão mais enxuta da AlexNet por possuir menos camadas e nós. Isso foi necessário pois o ambiente de execução foi realizado em torno de uma cpu com 4 processadores (em uma gpu seria muito mais rápido), logo, uma arquitetura mais simples era necessária para realizar as inferências.

Além disso, devemos citar que utilizamos a entropia-cruzada como função de perda em conjunto com o gradiente descendente.

Com relação ao treinamento ele utilizou 21 épocas de modo que iniciou-se a taxa de aprendizado em 0.1 e após algumas iterações uma desaceleração ocorria. Ao final da última execução tínhamos uma taxa de 0.001. Essa desaceleração foi necessária pois inicialmente precisamos que haja um aprendizado mais “rápido” dado a grande quantidade de pesos. Conforme há uma conversão, reduzimos a velocidade para que o algoritmo seja mais estável.

Normalização

As imagens do dataset cifar10 estão no intervalo [0,255], logo, uma normalização foi realizada para elas estivessem dentro de [0,1]. Essa etapa é necessária para que o algoritmo consiga convergir de forma mais adequada.

Teste

Dada a arquitetura descrita neste documento nós realizamos o aprendizado e avaliação no próprio conjunto do cifar10. Na avaliação foi possível obter uma acurácia de aproximadamente 75%.

Após isso realizamos um teste na finalidade de verificar a generalização do modelo em imagens que não fazem parte do conjunto cifar10. As figuras foram pesquisadas através do google sendo randomicamente selecionadas.

Figura 1: Imagens escolhidas para teste.

Com o teste realizado obtemos os seguintes resultados descritos na tabela:

Classe verdadeira	Classe predita	Probabilidade da classe
Airplane	Airplane	0.9981134
Automobile	Automobile	0.985059
Bird	Bird	0.9782802
Cat	Cat	0.9743745
Deer	Deer	0.9955754
Dog	Dog	0.68504757
Frog	Bird	0.45559782
Horse	Horse	0.9100185
Ship	Ship	0.9652143
Truck	Truck	0.96826696
Acurácia	0.9	

Tabela 1: Resultado do teste.

Conclusão

Com o resultado do teste vemos que há uma acurácia de 90% no conjunto de dados nunca visto, o que demonstra um razoável poder de generalização. Note que com exceção das classes Dog e Frog o estimador conseguiu determinar com uma boa precisão as classes.

Acreditamos que o fato destas classes não serem determinadas com clareza é por conta da arquitetura simples da rede. Como não possuímos muitas camadas e neurônios não é possível extrair muitas características do treinamento, logo, quando apresentamos uma imagem mais complexa para o modelo, fica difícil ele realizar uma inferência correta.

Confiamos, também, que essa precisão não é causada pelos exemplos do treinamento, visto que temos uma grande quantidade de amostras e as mesmas estão balanceadas.

Portanto, podemos concluir que mesmo com a simplicidade da rede ainda é possível gerar um classificador razoavelmente poderoso e com um esforço muito pequeno dado que o TensorFlow abstrai muito o trabalho do programador (código fonte possui apenas 124 linhas). Por fim, caso adicionássemos mais camadas e neurônios, certamente obteríamos uma classificador ainda melhor.