Convolutional Neural Nets + TensorFlow

Tóp. Esp. BD III
Christian Cardozo
Prof.: Geraldo Zimbrão

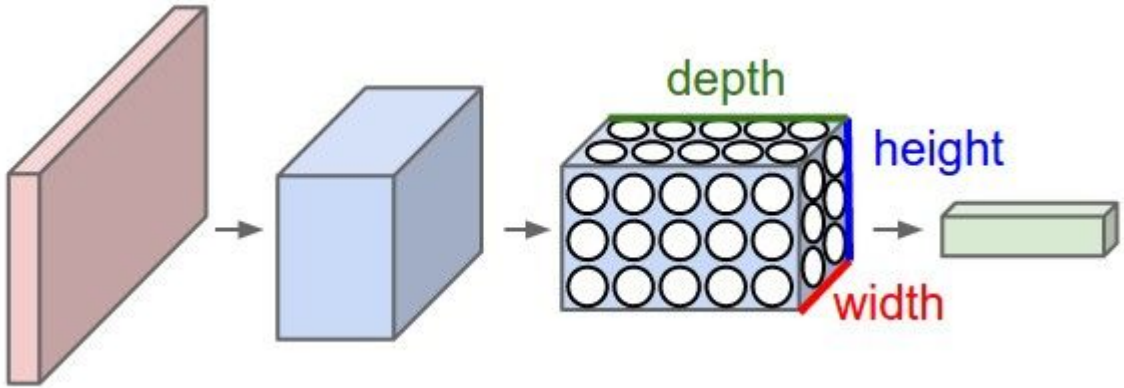
Convolutional Neural Network (CNN)

- Modelo de rede neural onde o input comumente é uma imagem
- Ex. de datasets.: MNIST, CIFAR-10, CIFAR-100, etc
- ConvNet organiza os neurônios em 3 dimensões.
- A última camada de uma ConvNet tem 1x1xn dimensões, onde n é o número de classes
- Para tratar imagens, uma rede neural tradicional possui um overhead de conexões
- Cada layer recebe um volume 3D como input e produz um outro volume 3D como output

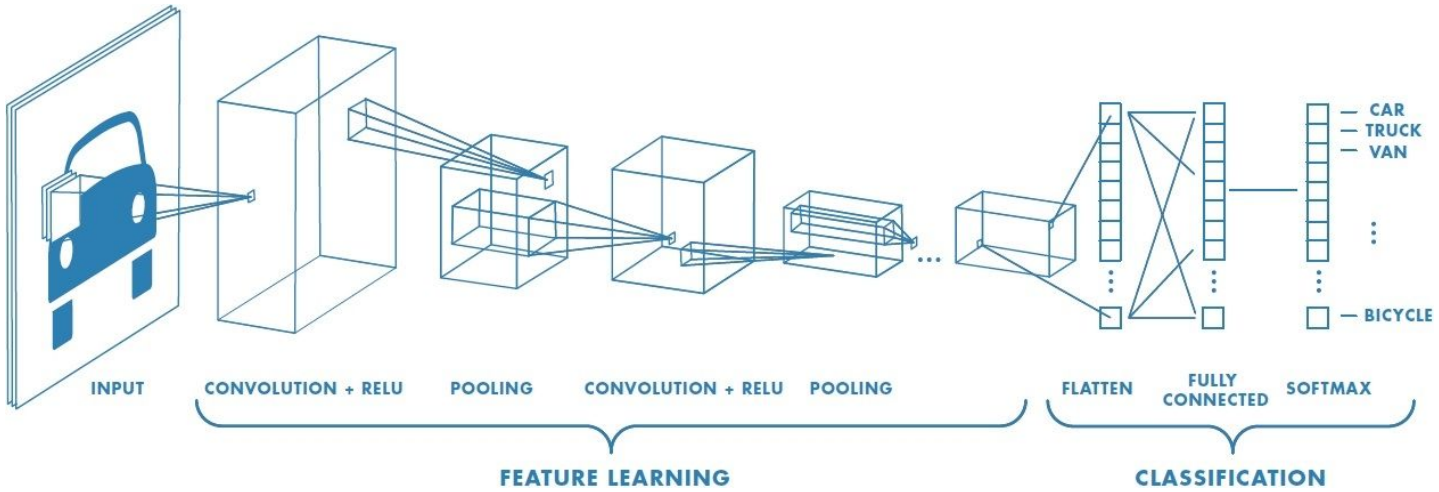
Convolutional Neural Network (CNN)

- ‘3’ tipos de camadas:
  - Convolutional layer (CONV)
  - Pooling layer (POOL)
  - Fully-connected layer (FC)
  - “ “ “(RELU Layer) “ “ “ (RELU)
- Layers com parâmetros: CONV e FC
- Layers sem parâmetros: RELU e POOL
- Layers com hiperparâmetros: CONV, FC e POOL
- Layers sem hiperparâmetros: RELU

Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)

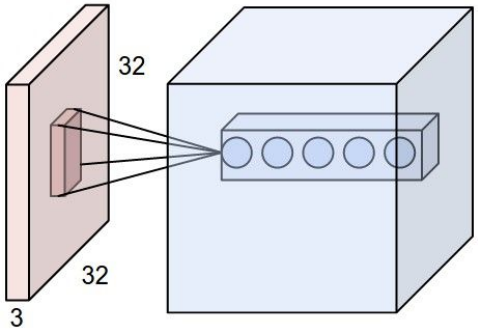


CNN - Convolutional Layer

- Aplica filtros na imagem de entrada
- Os filtros são matrizes, normalmente pequenas: 3x3, 5x5
    - Conectividade local: Cada neurônio da CONV layer enxerga apenas uma parte do input (isso reduz o número de conexões comparada à uma rede neural tradicional)
    - Repective Field é o hiperparam. que define o quanto cada neurônio vê: é equivalente ao tamanho do filtro.
- Em uma CONV layer, todos os neurônios com W e H dos diversos níveis de profundidade enxergam exatamente a mesma região do input - cada neurônio com seus pesos distintos.

CNN - Convolutional Layer

- Cada neurônio da CONV layer é conectado à uma localidade do input, seguindo a mesma regra “full depth”.



CNN - Convolutional Layer

- Parâmetros que controlam o tamanho do output
  - Depth: quantidade de filtros aplicados
    - Uma “depth column” também é chamado de fiber (fibra)
  - Stride: é a quantidade de slide que o filtro faz no input
  - Zero-padding: preenchimento das bordas com o valor zero
- Fórmula para quantos neurônios terão no output:
  - #neurons = ((W-F+2P)/S) + 1

CNN - Convolutional Layer

- Há restrições no valor de Stride. Ex.: se W = 10, P = 0 e F = 3, não é possível realizar S = 2
- Exemplo real de uma ConvNet:
  - Input 227x227x3 (3 canais: RGB)
  - F = 11, S = 4, P = 0, qtd filtros K = 96.
  - A saída será: (227 - 11 + 2*0)/4 + 1 = 55
  - 55x55x96
  - Neurônios de uma mesma “column depth” estão conectados à mesma região de tamanho 11x11x3 do input

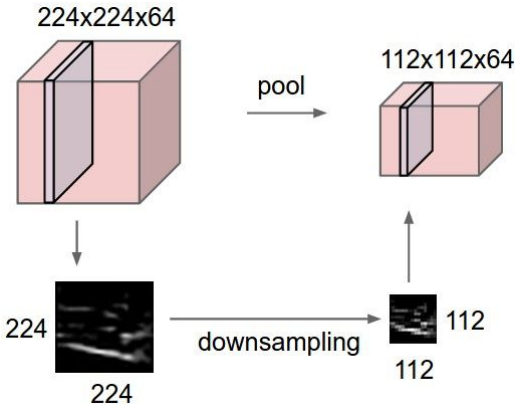
CNN - Convolutional Layer

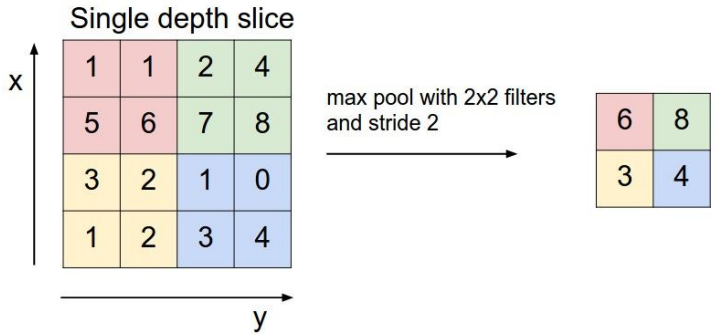
- Parameter Sharing: para reduzir a quantidade de parâmetros, todos os neurônios de um mesmo nível de profundidade usam o mesmo conjunto de pesos.
- Por filtro temos F * F * D1 pesos
- (F * F * D) * K + K pesos no total na camada (últ. soma são “+K” biases)
- “Magic Numbers”:
  - F = 3
  - S = 1
  - P = 1

CNN - Pooling Layer

- Estratégia para reduzir dimensionalidade (o depth permanece o mesmo)
- É aplicado de forma independente para cada nível de profundidade
- Aplica uma função (max, media, l2 norm) em pequenas regiões de cada nível de profundidade
- 2 hiperparâmetros: Stride (S) e percepção espacial (F)
- Uso de POOL é contestado por alguns papers (Striving for Simplicity: The All Convolutional Net, Springenberg J.T. et al)

CNN - Pooling Layer





CNN - Pooling Layer

- Realiza um “downsampling”
- Max tem se saído melhor na prática
- “Magic Numbers”:
  - F = 2 e S = 2
  - F = 3 e S = 2
  - Números maiores que isso deixa o POOL muito destrutivo

CNN - Fully-conected layer

- Igual às camadas das redes neurais tradicionais
- Neurônios conectados à todos valores de entrada
- Conversão FC ⇔ CONV é possível

CNN - Arquiteturas

- Normalmente seguem a regra:
  - INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
- Prefira várias CONV com pequenos filtros à uma CONV com filtro maior
- Patterns:
  - Input layer com dimensões divisíveis por 2 várias vezes
  - Filtros pequenos
  - Usar padding na primeira CONV para preservar a imagem toda
- Arquiteturas famosas: LeNet, AlexNet, VGGNet, etc.

TensorFlow

- An open-source software library for Machine Intelligence
- Unidade central é o **Tensor**
- Tensor é um array de tipos primitivos com ranks
    - 3 # a rank 0 tensor; this is a scalar with shape []
    - [1., 2., 3.] # a rank 1 tensor; this is a vector with shape [3]
    - [[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
    - [[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
- API de mais baixo nível: Tensor Core
- No TF, cria-se um grafo computacional para depois executá-lo. Cada nó recebe zero ou mais tensores de entrada e retorna um tensor como output

TensorFlow

- Para rodar o grafo, criamos uma sessão que encapsula os controles e estados
- Podemos combinar nós em operações (que também são nós)
- Exemplos de tipos de nó:
    - constant: valores fixos pré-inicializados
    - placeholders: parametriza a entrada de dados
    - variable: parâmetros treináveis nos modelos

TensorFlow - Instalação

- Guia: https://www.tensorflow.org/install/install_linux#InstallingNativePip
- Instalação do tensor flow no Ubuntu 16.04 sem suporte à GPU usando PIP:
  sudo apt-get install python3-pip python3-dev
  sudo pip3 install tensorflow
- Para ver se tudo funciona, rodar o seguinte código Python:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

- Se printar 'Hello, TensorFlow!', está tudo ok!

TensorFlow - Programas Exemplos

- *** abrir códigos de programas ***
  - add_and_multiply.py
  - linear_model.py
  - linear_model_perfect.py
  - linear_model_gradiente_descent.py
  - linear_model_gradiente_loss.py
  - estimator.py

TensorFlow - MNIST

- MNIST: dataset com números escritos à mão
  - 55 mil exemplos de treinamento
  - 10 mil exemplos de teste
  - 5 mil exemplos de validação
- Cada imagem tem dimensão 28x28x1 = 784 números
- Tentativa 1: Regressão Softmax (mnist_softmax_regression.py) => 92% (menos de 1 min)
- Tentativa 2: CNN (mnist_cnn.py) => 99,26% (em 40 min)
- Arquitetura da CNN usada:
  - INPUT -> CONV -> POOL -> CONV -> POOL -> FC -> FC

TensorFlow - CNN classes

- Tf.nn.conv2d
  - conv2d(input, filter ,strides, padding, use_cudnn_on_gpu=None, data_format=None, name=None)
  - input: X exemplos em 4D [batch, in_height, in_width, in_channels]
  - filter: W pesos em 4D [filter_height, filter_width, in_channels, out_channels]
  - strides: deve seguir o seguinte padrão: strides = [1, stride, stride, 1]
  - Padding: “SAME” ou “VALID”
- Retorna um 4D tensor do mesmo tipo do input
- Ref.: https://www.tensorflow.org/api_docs/python/tf/nn/conv2d

TensorFlow - CNN classes

- Tf.nn.relu
  - relu(features, name=None)
  - Computa max(feature, 0)
  - features: um tensor que deve ser de um dos seguintes tipos: float32, float64, int32, int64, uint8, int16, int8, uint16, half.
- Retorna um tensor do mesmo tipo de features
- Ref.: https://www.tensorflow.org/api_docs/python/tf/nn/relu

TensorFlow - CNN classes

- Tf.nn.max_pool
  - max_pool(value, ksize, strides, padding, data_format='NHWC', name=None)
  - value: um tensor 4D [batch, height, width, channels] de float32
  - Ksize: lista de ints com length >= 4. Tamanho da janela para cada dim do input
  - Strides: lista de ints com length >= 4. Tamanho do slide para cada dim do input
  - padding: “VALID” ou “SAME”