Contents

[Contents 1](#_Toc482100576)

[Deep learning framework 1](#_Toc482100577)

[Keras 2](#_Toc482100578)

[安装 3](#_Toc482100579)

[workflow 4](#_Toc482100580)

[tensorflow 5](#_Toc482100581)

[Machine learning 5](#_Toc482100582)

[MLP (multi-layer perception) 5](#_Toc482100583)

[deep learning 6](#_Toc482100584)

[Introduction to Deep Learning Algorithms 6](#_Toc482100585)

[一天搞懂深度学习 –李宏毅 7](#_Toc482100586)

[NN 7](#_Toc482100587)

[CNN 8](#_Toc482100588)

[RNN (Recurrent Neural Network) 9](#_Toc482100589)

[CS231n Convolutional Neural Networks for Visual Recognition 9](#_Toc482100590)

[NN (Neural Network) 9](#_Toc482100591)

[CNN (Convolutional Neural Networks / ConvNets) 13](#_Toc482100592)

[DeepLearning 17](#_Toc482100593)

# Deep learning framework

|  |  |  |  |
| --- | --- | --- | --- |
|  | By | interface | description |
| Caffe2 | the Berkeley Vision and Learning Center (BVLC) | C, C++, Python, MATLAB, cmd | deep learning framework  made with expression, speed, and modularity in mind |
| CNTK | The Microsoft Cognitive Toolkit | Python, C++, C# and cmd | unified deep-learning toolkit  train and combine popular model types across multiple GPUs and servers |
| TensorFlow | Google’s Machine Intelligence research organization | C++, Python | software library  numerical computation using data flow graphs |
| theano |  | Python | math expression compiler  defines, optimizes, and evaluates mathematical expressions |
| Torch |  | C, C++, Lua | scientific computing framework  offers wide support for machine learning algorithms. |
| Keras |  | Python | neural networks library  running on top of either TensorFlow or Theano, fast experimentation |

## Keras

<https://keras-cn.readthedocs.io/en/latest/>

Keras的核心数据结构是“模型”,Keras的底层库使用Theano或TensorFlow(“符号主义”的库)

与传统的Python代码区别？

符号主义的计算首先定义各种变量，然后建立“计算图”，计算图规定了各个变量之间的计算关系。建立好的计算图需要编译已确定其内部细节，然而，此时的计算图还是一个“空壳子”，里面没有任何实际的数据，只有当你把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值

深度学习的优化算法，说白了就是梯度下降。每次的参数更新有两种方式：

Batch gradient descent（批梯度下降）：遍历全部数据集算一次损失函数，然后算函数对各个参数的梯度，更新梯度。缺点：计算量开销大，计算速度慢，不支持在线学习

stochastic gradient descent（随机梯度下降）：速度快，但收敛性不好，可能在最优点附近晃来晃去，hit不到最优点。两次参数的更新也有可能互相抵消掉，造成目标函数震荡的比较剧烈

mini-batch gradient decent（小批的梯度下降）：数据分为若干批，按批来更新参数，批中的一组数据共同决定了本次梯度的方向，下降起来就不容易跑偏，减少了随机性

张量可以看作是向量、矩阵的自然推广，我们用张量来表示广泛的数据类型

0阶张量，即标量，也就是一个数

1阶张量，也就是一个向量

2阶张量，也就是一个矩阵

3阶张量，一个立方体

张量的阶数有时候也称为维度，或者轴

Ubuntu 16.04 LTS是Nvidia官方以及绝大多数深度学习框架默认开发环境

Theano: a compiler for mathematical expressions in Python

TensorFlow: a python library for fast numerical computing

Keras: library addresses these concerns by providing a wrapper for both Theano and Tensorflow

scikit-learn library：general purpose machine learning framework in Python built on top

of SciPy

application checkpointing

dropout

convolutional neural networks

### 安装

# 系统升级

$ sudo apt update

$ sudo apt upgrade

# 安装python基础开发包

$ sudo apt install -y python-dev python-pip python-nose gcc g++ git gfortran vim

# 安装运算加速库

$ sudo apt install -y libopenblas-dev liblapack-dev libatlas-base-dev

# 安装CUDA开发环境

$ sudo dpkg -i cuda-repo-ubuntu1604-8-0-local\_8.0.44-1\_amd64.deb

$ sudo apt update

$ sudo apt install cuda

# 将CUDA路径添加至环境变量

$ sudo gedit /etc/bash.bashrc

在bash.bashrc文件中添加：

export CUDA\_HOME=/usr/local/cuda-8.0

export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}

export LD\_LIBRARY\_PATH=/usr/local/cuda-8.0/lib64${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

$ source gedit /etc/.bashrc

在.bashrc中添加如上相同内容

$ sudo gedit ~/.bashrc

$ nvcc -v 测试nVidia cuda版本号

Keras框架搭建

$ sudo pip install -U --pre pip setuptools wheel

$ sudo pip install -U --pre numpy scipy matplotlib scikit-learn scikit-image

$ sudo pip install -U --pre theano

$ sudo pip install -U --pre keras

注：依赖numpy，scipy, pyyaml, HDF5, h5py（可选，仅在模型的save/load函数中使用）

TensorFlow(当使用TensorFlow为后端时) or Theano(当使用Theano作为后端时), Keras默认使用TensorFlow作为后端来进行张量操作

keras$ sudo python setup.py install

or $ sudo pip install keras

$ python 验证

>>> import theano

>>> import keras

Keras环境设置

修改默认keras后端: gedit ~/.keras/keras.json

配置theano文件: gedit ~/.theanorc

[global]

openmp=False

device = gpu

floatX = float32

allow\_input\_downcast=True

[lib]

cnmem = 0.8

[blas]

ldflags= -lopenblas

[nvcc]

fastmath = True

验证keras是否安装成功

>>>import keras

加速测试

keras/examples/$ python mnist\_mlp.py

### 概念

#### 函数式（Functional）模型

from keras.layers import Input, Dense

from keras.models import Model

# This returns a tensor

inputs = Input(shape=(784,))

**层对象接受张量为参数，返回一个张量。**

# a layer instance is callable on a tensor, and returns a tensor

x = Dense(64, activation='relu')(inputs)

x = Dense(64, activation='relu')(x)

predictions = Dense(10, activation='softmax')(x)

**输入是张量，输出也是张量的一个框架就是一个模型，通过Model定义。**

# This creates a model that includes the Input layer and three Dense layers

model = Model(inputs=inputs, outputs=predictions)

model.compile(optimizer='rmsprop',loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(data, labels) # starts training

**重用已经训练好的模型**

把模型当作一个层一样，通过提供一个tensor来调用它。注意当你调用一个模型时，你不仅仅重用了它的结构，也重用了它的权重。

x = Input(shape=(784,))

# This works, and returns the 10-way softmax we defined above.

y = model(x)

**多输入和多输出模型**

main\_input = Input(shape=(100,), dtype='int32', name='main\_input')

...

auxiliary\_output = Dense(1, activation='sigmoid', name='aux\_output')(main\_medi)

auxiliary\_input = Input(shape=(5,), name='aux\_input')

x = keras.layers.concatenate([main\_medi, auxiliary\_input])

...

main\_output = Dense(1, activation='sigmoid', name='main\_output')(x)

model = Model(inputs=[main\_input, auxiliary\_input], outputs=[main\_output, auxiliary\_output])

model.compile(optimizer='rmsprop',

loss={'main\_output': 'binary\_crossentropy', 'aux\_output': 'binary\_crossentropy'},

loss\_weights={'main\_output': 1., 'aux\_output': 0.2})

# And trained it via:

model.fit({'main\_input': headline\_data, 'aux\_input': additional\_data},

{'main\_output': labels, 'aux\_output': labels},

epochs=50, batch\_size=32)

**共享层**

把一个相同的Conv2D应用于一个大小为(3,32,32)的数据，然后又将其应用于一个(3,64,64)的数据，那么此时该层就具有了多个输入和输出的shape

a = Input(shape=(3, 32, 32))

b = Input(shape=(3, 64, 64))

conv = Conv2D(16, (3, 3), padding='same')

conved\_a = conv(a)

# Only one input so far, the following will work:

assert conv.input\_shape == (None, 3, 32, 32)

conved\_b = conv(b)

# now the `.input\_shape` property wouldn't work, but this does:

assert conv.get\_input\_shape\_at(0) == (None, 3, 32, 32)

assert conv.get\_input\_shape\_at(1) == (None, 3, 64, 64)

Keras模型

model.summary()：打印出模型概况

config = model.get\_config():返回包含模型配置信息的Python字典。

model = Model.from\_config(config): 模型也可以从它的config信息中重构回去

model.get\_layer()：依据层名或下标获得层对象

model.get\_weights()：返回模型权重张量的列表，类型为numpy array

model.set\_weights()：从numpy array里将权重载入给模型，要求数组具有与model.get\_weights()相同的形状。

json\_string =model.to\_json：返回代表模型的JSON字符串，仅包含网络结构，不包含权值。

model = model\_from\_json(json\_string): 可以从JSON字符串中重构原模型

model.save\_weights(filepath)：将模型权重保存到指定路径，文件类型是HDF5（后缀是.h5）

model.load\_weights(filepath, by\_name=False)：从HDF5文件中加载权重到当前模型中, 默认情况下模型的结构将保持不变。如果想将权重载入不同的模型（有些层相同）中，则设置by\_name=True，只有名字匹配的层才会载入权重

Sequential模型

model.layers: 是添加到模型上的层的list

add(self, layer): 向模型中添加一个层, Layer对象

compile(self, optimizer, loss, metrics=None, sample\_weight\_mode=None)

optimizer：优化器对象

loss：目标函数

metrics：列表，包含评估模型在训练和测试时的网络性能的指标，典型用法是metrics=['accuracy']

### workflow

Load Data.

Define Model.

Compile Model.

Fit Model.

Evaluate Model.

Tie It All Together.

example<http://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks>

lung segmentation

<https://blog.altoros.com/experimenting-with-deep-neural-networks-for-x-ray-image-segmentation.html>

Melanoma (lesion analysis)

<https://isic-archive.com/>

[**https://github.com/iamrosmarin/BSc\_Thesis\_Skin\_Lesion\_Detection**](https://github.com/iamrosmarin/BSc_Thesis_Skin_Lesion_Detection)

## tensorflow

tensor (n维数组)　+ 计算图

充分利用CPU and GPU

# Machine learning

## MLP (multi-layer perception)

Sigmoid + Sigmoid + mse

batch Gradient decent, mini-batch gradient decent, SGD

<http://iamtrask.github.io/2015/07/12/basic-python-network/> <http://iamtrask.github.io/2015/07/27/python-network-part2/>

<https://iamtrask.github.io/2015/07/27/python-network-part2/>

drops out<http://iamtrask.github.io/2015/07/28/dropout/>



The line represents the error the network generates for every value of a particular weight, The balls in the picture signify various weights, The ball's initial positions are randomly generated, If two balls randomly start in the same colored zone, they will converge to the same point. This makes them redundant! They're wasting computation and memory! This is exactly what happens in neural networks.

Why Dropout: Dropout helps prevent weights from converging to identical positions. It does this by **randomly turning nodes off when forward propagating**. It then **back-propagates with all the nodes turned on**

if(do\_dropout):

layer\_1 \*= np.random.binomial([np.ones((len(X),hidden\_dim))],1-dropout\_percent)[0]

\* (1.0/(1-dropout\_percent))

if you're turning off half of your hidden layer, you want to double the values that ARE pushing forward so that the output compensates correctly

Dropout during training. not on your testing dataset

Tanh + softmax + cross-entropy loss

<http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/>

动量项

<http://blog.csdn.net/bvl10101111/article/details/54973284>

batch normalization

xk = xk – E[xk] / sqrt(Var[xk])

yk = rkxk + betak

对mini-batch数据集，沿着第k维进行batch normalization(公式如上)

优点：

improves gradient flow through the networks

allows higher learning rates

reduces the strong dependence on initialization

acts as a form of regularization, slightly reduces the need for dropout

role of bias in Neural Networks

sigmoid(wx+b), w改变的是sigmoid曲线的steepness, b改变的是将x平移

# deep learning

## Introduction to Deep Learning Algorithms

Traditional feedforward neural networks can be considered to have depth equal to the number of layers (i.e. the number of hidden layers plus 1, for the output layer). Support Vector Machines (SVMs) have depth 2 (one for the kernel outputs or for the feature space, and one for the linear combination producing the output).

Motivations for Deep Architectures

1. Insufficient depth can hurt

Depth 2 is enough in many cases (e.g. logical gates, formal [threshold] neurons, sigmoid-neurons, Radial Basis Function [RBF] units like in SVMs) to represent any function with a given target accuracy. But this may come with a price: that the required number of nodes in the graph may grow very large.

1. The brain has a deep architecture
2. Cognitive processes seem deep

Humans organize their ideas and concepts hierarchically.

Humans first learn simpler concepts and then compose them to represent more abstract ones.

Engineers break-up solutions into multiple levels of abstraction and processing

Deep Learning Tutorial

Training set

Validation set (perform model selection and hyper-parameter selection)

Test set (evaluate the final generalization error and compare different algorithms)

## 一天搞懂深度学习 –李宏毅http://www.slideshare.net/tw\_dsconf/ss-62245351

### NN



A two layers of logic gates can represent **any Boolean function**

A hidden layer network can represent **any continuous function**

Using multiple layers of neurons to represent some functions are much simpler -> less parameters, less data?

直接分类长发女，短发女，长发男，短发男，很困难，因为训练集少。但若先分类男女？长短头发？由这两个basic classifier再组合进一步分类，更容易

 

Learning Rates

小梯度，平坦，则大步伐；大梯度，陡峭，则小步伐

 

Weight Decay

Our brain prunes out the useless link between neurons, doing the same thing to machine’s brain improves the performance

Dropout直观理解：

 

 

### CNN

Why CNN for Image?

Some patterns are much smaller than the whole image

Subsmapling the piexls will not change the object

 



### RNN (Recurrent Neural Network)

## CS231n Convolutional Neural Networks for Visual Recognition

### NN (Neural Network)

网络权值的解释？W = 模板

Interpretation of **linear classifiers as template matching**. Another interpretation for the weights W is that each row of W corresponds to a template (or sometimes also called a prototype) for one of the classes. The score of each class for an image is then obtained by comparing each template with the image using an inner product (or dot product) one by one to find the one that “fits” best. With this terminology, the linear classifier is doing template matching, where the templates are learned. Another way to think of it is that we are still effectively doing Nearest Neighbor, but instead of having thousands of training images we are only using a single image per class (although we will learn it, and it does not necessarily have to be one of the images in the training set), and we use the (negative) inner product as the distance instead of the L1 or L2 distance.

for example: the ship template (W) contains a lot of blue pixels as expected. This template will therefore give a high score once it is matched against images of ships on the ocean with an inner product.

Additionally, note that the horse template seems to contain a two-headed horse, which is due to both left and right facing horses in the dataset. The linear classifier merges these two modes of horses in the data into a single template. Similarly, **the car linear classifier seems to have merged several modes into a single template which has to identify cars from all sides, and of all colors**. In particular, this template ended up being red, which hints that there are more red cars in the CIFAR-10 dataset than of any other color. The linear classifier is too weak to properly account for different-colored cars, but as we will see later neural networks will allow us to perform this task. Looking ahead a bit, a **neural network will be able to develop intermediate neurons in its hidden layers that could detect specific car types (e.g. green car facing left, blue car facing front, etc.), and neurons on the next layer could combine these into a more accurate car** score through a weighted sum of the individual car detectors.

First-layer Visualizations



Examples of visualized weights for the first layer of a neural network. Left: **Noisy features indicate could be a symptom: Unconverged network, improperly set learning rate, very low weight regularization penalty**. Right: Nice, smooth, clean and diverse features are a good indication that the training is proceeding well.

Multi-layer Visualization

Layer1 focus on **local domain feature**

Layer2 combine layer1’s output (as basic classifier) = combine local feature to **multi-features**



linear mapping -> Neural Networks -> Convolutional Neural Networks

score function: mapping the raw image pixels to class scores (e.g. a linear function)

loss(cost) function: measured the quality of a particular set of parameters based on how well the induced scores agreed with the ground truth labels in the training data (e.g. Softmax/SVM).

SVM cost function is an example of a convex function --- convex optimization

Neural Networks cost functions will become non-convex

如何优化？

Optimization: the process of finding the set of parameters W that minimize the loss function.

1. Random search (bad idea)

W = np.random.randn(10, 3073) \* 0.0001 # generate random parameters

1. Random Local Search

Core idea: iterative refinement, refining a specific set of weights W to be slightly better is significantly less difficult

Wtry = W + np.random.randn(10, 3073) \* step\_size

1. Following the Gradient

weights += - step\_size \* weights\_grad # perform parameter update

The gradient tells us the direction in which the function has the steepest rate of increase, but it does not tell us how far along this direction we should step

choosing the step size (also called the learning rate) will become one of the most important (and most headache-inducing) hyper parameter settings in training a neural network

如何计算梯度？

Computing the gradient: numerical gradient and analytic gradient (e.g. **chain rule == backpropagation**)

Backpropagation allow us to efficiently optimize relatively arbitrary loss functions that express all kinds of Neural Networks, including Convolutional Neural Networks.

Batch梯度下降法(BGD)：it seems wasteful to compute the full loss function over the entire training set in order to perform only a single parameter update

Mini-batch 梯度下降法(MGD)：compute the gradient over batches of the training data

假定120万张图像，由1000 labels组成，那么120万张的平均data loss与1000张等值，the gradient from a mini-batch is a good approximation of the gradient of the full objective (cost)

The size of the mini-batch is a hyper parameter but it is not very common to cross-validate it. It is usually based on memory constraints (if any), or set to some value, e.g. 32, 64 or 128. We use powers of 2 in practice because many vectorized operation implementations work faster when their inputs are sized in powers of 2.

Stochastic Gradient Descent (SGD) : on-line gradient descent

Tips:

if loss barely changing means Learning rate is probably too low

mini-batch size: 256

step size: 10\*\*(-5) (ideally, h->0)

2% improvement: train multiple independent models, at test time average their results

Mini-batch SGD

Loop:

1. Sample a batch of data
2. Forward prop it through the graph, get loss
3. Backprop to calculate the gradients
4. Update the parameters using the gradient

Regularization (dropout)

Randomly set some neurons to zero in the forward pass



解释1：



解释2:

Dropout is training a large ensemble of models (that share parameters). Each binary mask is one model gets trained on only one data point

Ideally, want to integrate out all the noise.

### CNN (Convolutional Neural Networks / ConvNets)

ConvNet architectures:

Images -> Convolutional Layer -> Pooling Layer -> Fully-Connected Layer -> output labels

ConvNet architectures make the explicit assumption that the inputs are images

Regular Neural Nets don’t scale well to full images, for example, image 200\*200\*3 -> 120,000 weights

full connectivity is wasteful and **the huge number of parameters would quickly lead to overfitting**.

3D volumes of neurons: unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.

Intuitively, the network will **learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer**, or eventually entire honeycomb or wheel-like patterns on higher layers of the network, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.



For example, if the first Convolutional Layer takes as input the raw image, then different neurons may activate in presence of various oriented edged, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a depth column (some people also prefer the term fibre).

Real-world example

the input volume size (W1 \* H1 \* D1): 227\*227\*3

the receptive field size of the Conv Layer neurons (F): 11

the number of filters (K): 96

the stride with which they are applied (S): 4

the amount of zero padding used (P) on the border: 0

the spatial size of the output volume (W2 \* H2 \* D2): 55\*55\*96

W2 = **(W1-F+2P)/S+1** = 55

H2 = (H1−F+2P)/S+1 = 55

D2 = K = 96

With parameter sharing, each filter has F\*F\*D1 (11\*11\*3 = 363) weights, **total weights (F\*F\*D1)\*K (363x96) and K biases**

理论上：参数有55x55x96 \* 11\*11\*3 多个

假设：Notice that **the parameter sharing assumption** is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images.

假设不一定对：Note that sometimes the parameter sharing assumption may not make sense. This is especially the case when the input images to a ConvNet have some specific centered structure, where we should expect, for example, that completely different features should be learned on one side of the image than another. One practical example is when the input are faces that have been centered in the image. You might expect that different eye-specific or hair-specific features could (and should) be learned in different spatial locations. In that case it is common to relax the parameter sharing scheme, and instead simply call the layer a Locally-Connected Layer.

若假设对，则图像空间共用同一个filter（理由如上）,则参数：96 \* 11\*11\*3

Backpropagation: The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters).

all 55\*55 neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

Pooling Layer



reduce the amount of parameters and computation in the network, and hence to also control overfitting

how? filters of size 2x2 applied with a stride of 2 down samples every depth slice

input volumn size: W1\*H1\*D1

filter (F): 2\*2

stride (S): 2

output volumn size: W2\*H2\*D2

W2 = **(W1-F)/S + 1**

H2 = (H1-F)/S + 1

D2 = D1

general pooling: max, average, L2-norm pooling

Backpropagation: Recall from the backpropagation chapter that **the backward pass for a max(x, y) operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass**. Hence, during the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called the switches) so that gradient routing is efficient during backpropagation.

Fully-connected layer

Converting FC layers to CONV layers

input volume of size: 7\*7\*512

F = 7

P = 0

S = 1

K = 4096

output: 1\*1\*4096

Layer Patterns

INPUT -> [[CONV -> RELU]\*N -> POOL?]\*M -> [FC -> RELU]\*K -> FC

usually

**0 <= N <= 3**

**0 <= M**

**0 <= K < 3**

INPUT -> FC, implements a linear classifier. Here N = M = K = 0.

INPUT -> CONV -> RELU -> FC

INPUT -> [CONV -> RELU -> POOL]\*2 -> FC -> RELU -> FC. Here we see that there is a single CONV layer between every POOL layer.

INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]\*3 -> [FC -> RELU]\*2 -> FC Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because **multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation**

多个小filter级别优于大filter

Suppose that you stack three 3x3 CONV layers on top of each other (with non-linearities in between, of course). In this arrangement, **each neuron on the first CONV layer has a 3x3 view of the input volume**. **A neuron on the second CONV layer has a 3x3 view of the first CONV layer, and hence by extension a 5x5 view of the input volume**. Similarly, a neuron on the third CONV layer has a 3x3 view of the 2nd CONV layer, and hence a 7x7 view of the input volume. Suppose that instead of these three layers of 3x3 CONV, we only wanted to use a single CONV layer with 7x7 receptive fields.

3个3\*3filter等效于7\*7filter，但有如下优点

First, the neurons would be computing a linear function over the input, while the three **stacks of CONV layers contain non-linearities that make their features more expressive**.

Second, if we suppose that all the volumes have C channels, then it can be seen that the single 7x7 CONV layer would contain C×(7×7×C)=49C2parameters, while the three 3x3 CONV layers would only contain 3×(C×(3×3×C))=27C2 parameters.

Intuitively, **stacking CONV layers with tiny filters as opposed to having one CONV layer with big filters allows us to express more powerful features of the input, and with fewer parameters**

INPUT: [224x224x3] memory: 224\*224\*3=150K weights: 0

CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M weights: (3\*3\*3)\*64 = 1,728

CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M weights: (3\*3\*64)\*64 = 36,864

POOL2: [112x112x64] memory: 112\*112\*64=800K weights: 0

CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M weights: (3\*3\*64)\*128 = 73,728

CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M weights: (3\*3\*128)\*128 = 147,456

POOL2: [56x56x128] memory: 56\*56\*128=400K weights: 0

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*128)\*256 = 294,912

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*256)\*256 = 589,824

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*256)\*256 = 589,824

POOL2: [28x28x256] memory: 28\*28\*256=200K weights: 0

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*256)\*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*512)\*512 = 2,359,296

POOL2: [14x14x512] memory: 14\*14\*512=100K weights: 0

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

POOL2: [7x7x512] memory: 7\*7\*512=25K weights: 0

FC: [1x1x4096] memory: 4096 weights: 7\*7\*512\*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 weights: 4096\*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 weights: 4096\*1000 = 4,096,000

TOTAL memory: 24M \* 4 bytes ~= 93MB / image (only forward! ~\*2 for bwd)

TOTAL params: 138M parameters

## DeepLearning

许多数学对象可通过将它们分解成多个组成部分，如整数可以质因数分解，矩阵可以奇异值分解，实对称矩阵可以特征分解