资料

书籍教程

1. Deep Learning by Yoshua Bengio, Ian Goodfellow and Aaron Courville, 2016. （已下载）
2. Neural Networks and Deep Learning by Michael Nielsen, 2015.
3. Deep Learning: Methods and Applications by Microsoft Research, 2014.5. （已下载）
4. Deep Learning Tutorial by LISA lab, University of Montreal, 2015.9. （已下载）
5. UFLDL old 教程 by Andrew Ng, 2013.
6. UFLDL new Tutorial by Andrew Ng
7. Hacker's guide to Neural Networks by Andrej Karpathy （博客未完结，已下载）

视频课程

1. CS224d: Deep Learning for Natural Language Processing by Richard Socher
2. Machine Learning by Andrew Ng in Coursera. 感觉Coursera上的课比录制的现场课容易些
3. Neural Networks for Machine Learning by Geoffrey Hinton in Coursera
4. Neural networks class by Hugo Larochelle from Université de Sherbrooke
5. Deep Learning Course by CILVR lab @ NYU

博客论坛

1. deeplearning.net 最全的深度学习资料网站
2. Best Paper Awards in Computer Science (since 1996)
3. hjimce 持续跟进最新理论，总结得很详细
4. zouxy09 主要是深度学习和机器学习的基础原理
5. pluskid 2013-现今 pluskid 2009-2012 机器学习和深度学习，个别理论很详细
6. tornadomeet 深度学习和机器学习，理论总结得很好，比较全
7. 算法组 论坛 综合论坛，部分帖子质量高

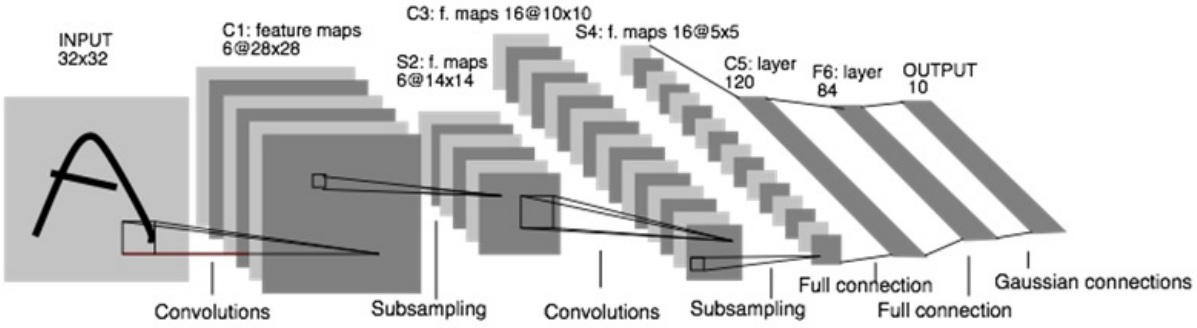
经典模型

卷积神经网络 (Convolutional Neural Network, CNN)

在ILSVRC的ImageNet数据集合上，人眼的辨识错误率大概在5.1%。主要的几种CNN模型在ILSVRC上的结果

模型名	AlexNet	VGG	GoogLeNet	ResNet
初入江湖	2012	2014	2014	2015
层数	8	19	22	152
Top-5错误	16.4%	7.3%	6.7%	3.57%
Data Augmentation	+	+	+	+
Inception(NIN)	-	-	+	-
卷积层数	5	16	21	151
卷积核大小	11,5,3	3	7,1,3,5	7,1,3,5
全连接层数	3	3	1	1
全连接层大小	4096,4096,1000	4096,4096,1000	1000	1000
Dropout	+	+	+	+
Local Response Normalization	+	-	+	-
Batch Normalization	-	-	-	+

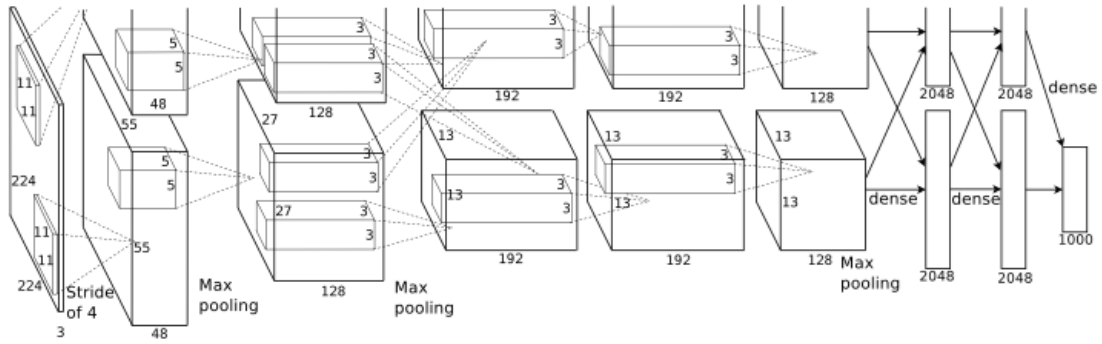
LeNet-5 (1998)



Papers:
Backpropagation Applied to Handwritten Zip Code Recognition (LeCun-1989)
Gradient-Based Learning Applied to Document Recognition(LeCun-1998)

LeNet用于手写数字的识别，在LeCun的主页有模型的完整介绍。

AlexNet (2012)



Paper:
ImageNet Classification with Deep Convolutional Neural Networks(Alex Krizhevsky-NIPS2012)

AlexNet相比传统的CNN（比如LeNet），提出了一些重要的修改：

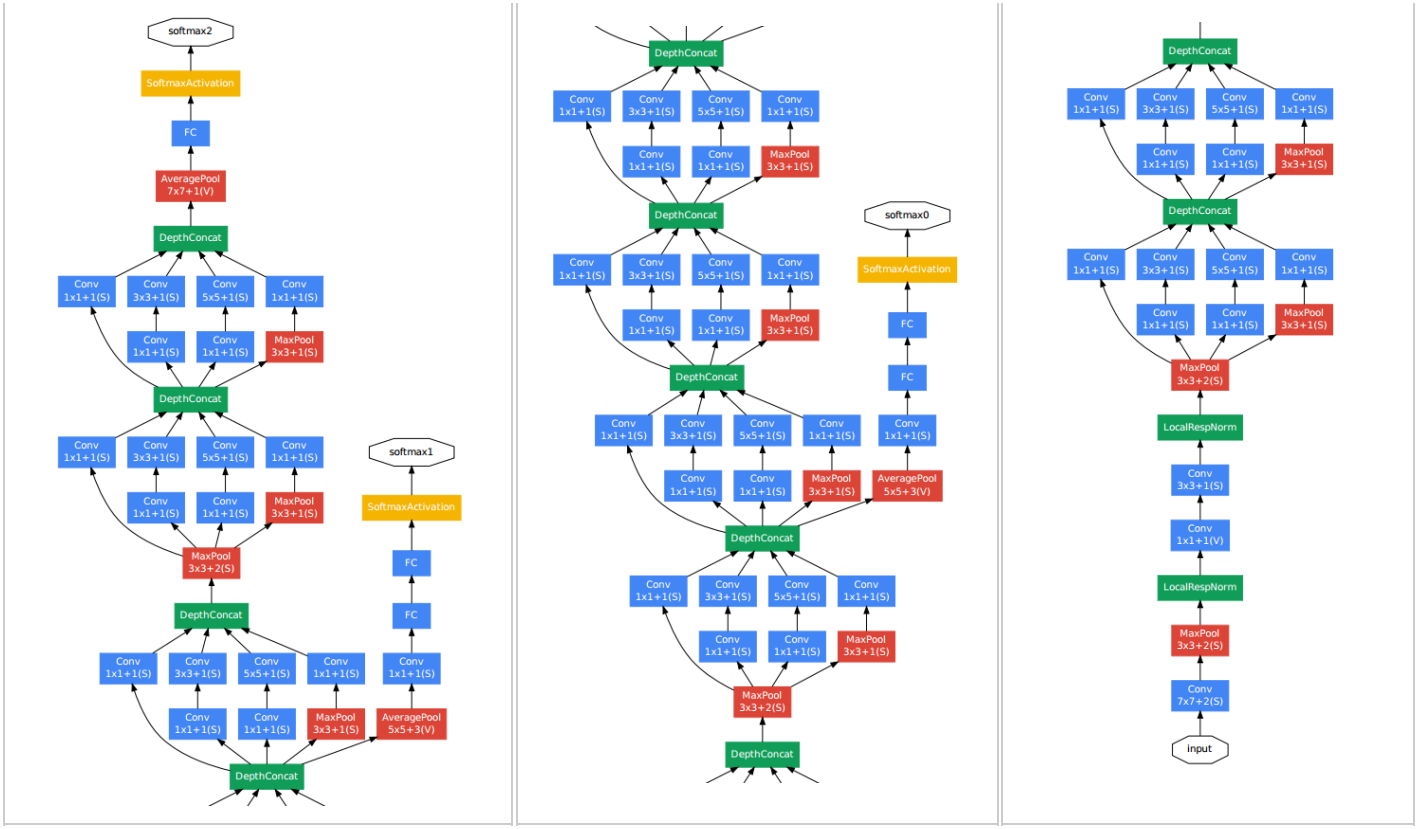
1. Data Augmentation
2. Dropout
3. ReLU
4. Local Response Normalization
5. Overlapping Pooling
6. 多GPU并行

VGG Net (2014)

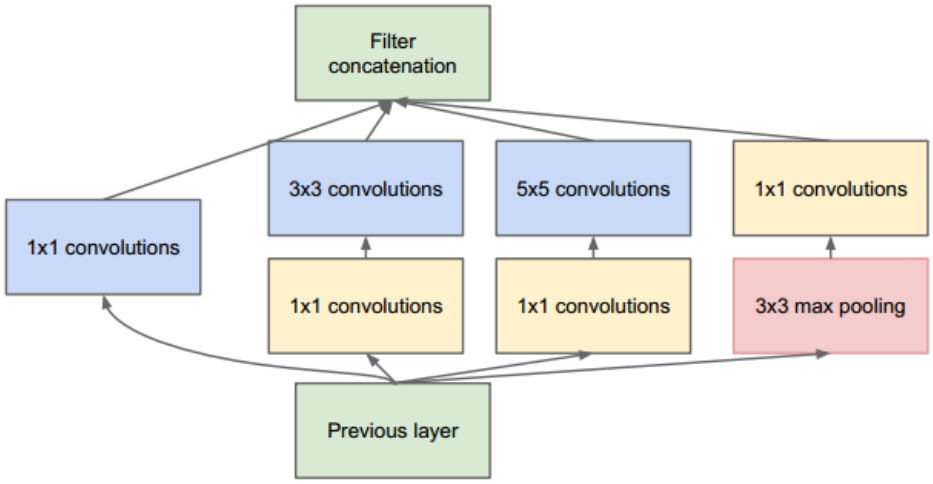
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Paper:
Very Deep Convolutional Networks for Large-Scale Image Recognition(K Simonyan-CVPR2014)
Homepage：http://www.robots.ox.ac.uk/~vgg/research/very_deep/

Google Net (2014)



主要的创新在于他的Inception，这是一种网中网（Network In Network）的结构，即原来的结点也是一个网络。Inception一直在不断发展，目前已经V2、V3、V4。Inception的结构如下图所示，其中1*1卷积主要用来降维，用了Inception之后整个网络结构的宽度和深度都可扩大，能够带来2-3倍的性能提升。



- [v1 2014] Going Deeper with Convolutions, 6.67% test error
  Inception v1的网络，将1x1，3x3，5x5的conv和3x3的pooling，stack在一起，一方面增加了网络的width，另一方面增加了网络对尺度的适应性；

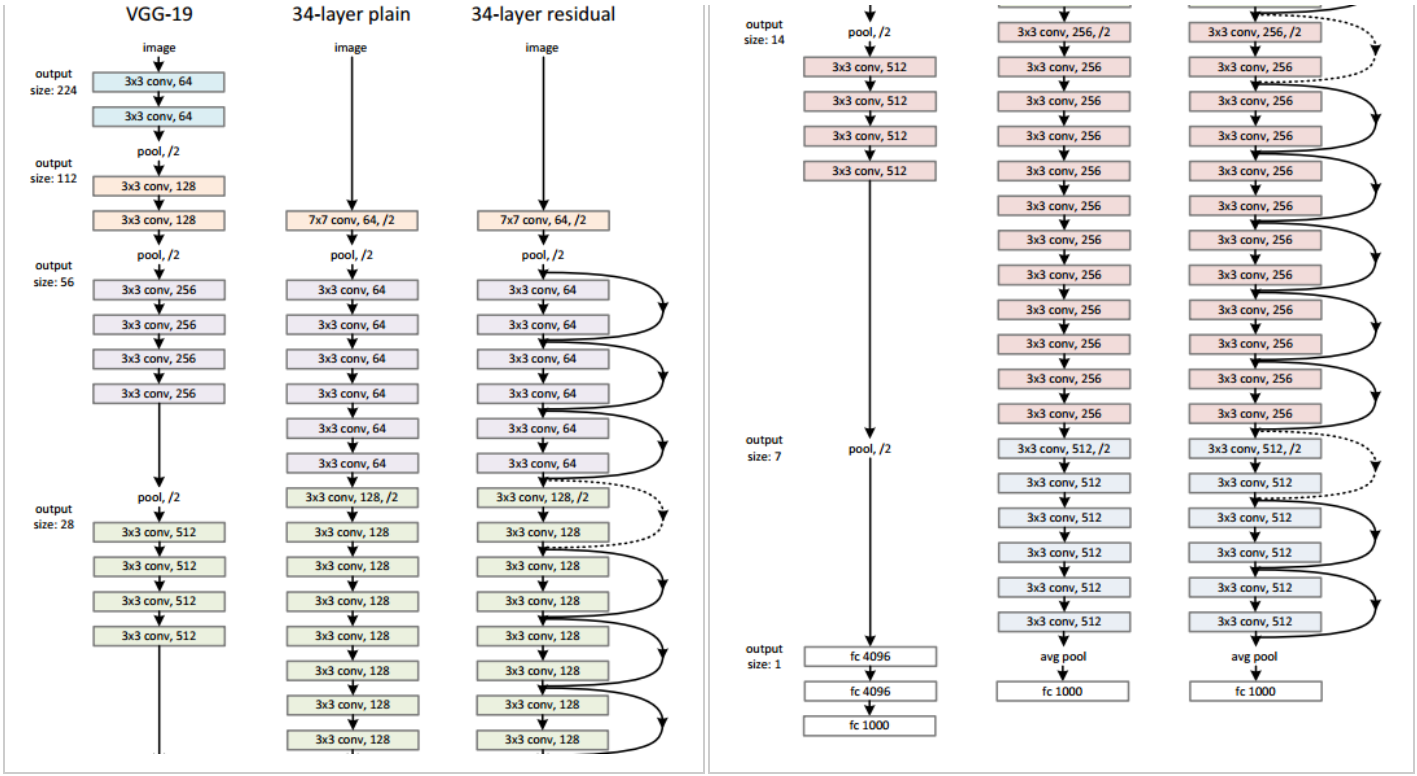
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

- [v2] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 4.8% test error
  v2的网络在v1的基础上，进行了改进，一方面了加入了BN层，减少了Internal Covariate Shift（内部neuron的数据分布发生变化），使每一层的输出都规范化到一个N(0, 1)的高斯，另外一方面学习VGG用2个3x3的conv替代inception模块中的5x5，既降低了参数数量，也加速计算；

- [v3] Rethinking the Inception Architecture for Computer Vision, 3.5% test error
  v3一个最重要的改进是分解（Factorization），将7x7分解成两个一维的卷积（1x7,7x1），3x3也是一样（1x3,3x1），这样的好处，既可以加速计算（多余的计算能力可以用来加深网络），又可以将1个conv拆成2个conv，使得网络深度进一步增加，增加了网络的非线性，还有值得注意的地方是网络输入从224x224变为了299x299，更加精细设计了35x35/17x17/8x8的模块；

- [v4] Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, 3.08% test error
  v4研究了Inception模块结合Residual Connection能不能有改进？发现ResNet的结构可以极大地加速训练，同时性能也有提升，得到一个Inception-ResNet v2网络，同时还设计了一个更深更优化的Inception v4模型，能达到与Inception-ResNet v2相媲美的性能。

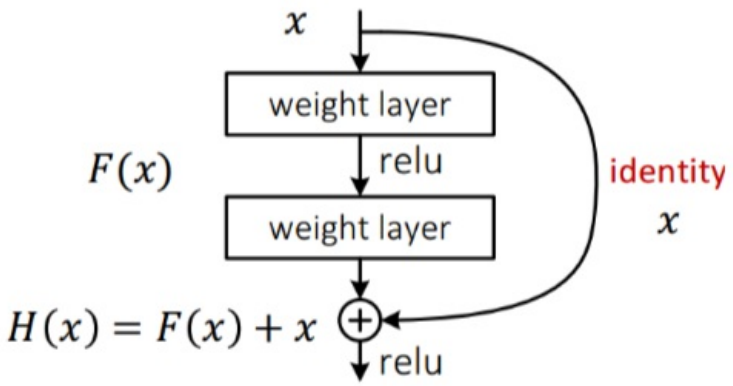
Residual Net (2015)



Paper:
Deep Residual Learning for Image Recognition(Kaiming He-CVPR2015)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

主要的创新在残差网络，如下图所示，其实这个网络的提出本质上还是要解决层次比较深的时候无法训练的问题。这种借鉴了Highway Network思想的网络相当于旁边专门开个通道使得输入可以直达输出，而优化的目标由原来的拟合输出H(x)变成输出和输入的差H(x)-x，其中H(X)是某一层原始的的期望映射输出，x是输入。



注意虚线部分均处于维度增加部分，亦即卷积核数目倍增的过程，这时进行`F(x) + x`就会出现二者维度不匹配，这里论文中采用两种方法解决这一问

题(其实是三种，但通过实验发现第三种方法会使performance急剧下降，故不采用):

- zero_padding: 对恒等层进行0填充的方式将维度补充完整。这种方法不会增加额外的参数
- projection: 在恒等层采用1x1的卷积核来增加维度。这种方法会增加额外的参数

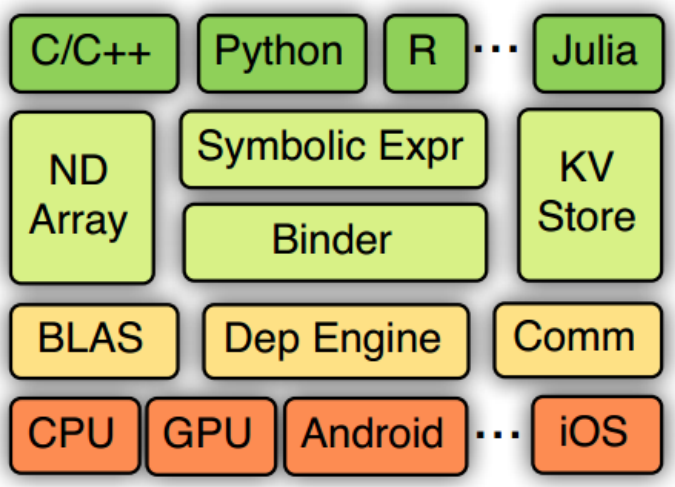
框架

MXNet

main site & doc: http://mxnet.io/
中文文档：https://github.com/dmlc/mxnet/tree/master/docs/zh
github: https://github.com/dmlc/mxnet
dmlc site: http://dmlc.ml/ github: https://github.com/dmlc
Model Zoo: https://github.com/dmlc/mxnet-model-gallery
examples: https://github.com/dmlc/mxnet/blob/master/example

概述

MXNet(mix-net) 使用C++编写，支持命令式和符号式设计，支持多种语言（C++, Python, R, Scala, Julia），支持多种平台（云，集群，物理机，手持设备）。MXNet 支持多卡和多机的并行。总体设计如图：



- Engine: 自动检测命令式和符号式代码中的数据依赖，并有效调度。命令式设计更灵活，易于debug，符号式设计易于全局优化。
- Symbol: 符号式设计接口。包含基本操作和卷积等复杂操作，支持符号自动优化和前向、后向计算。
- NDArray: 命令式的张量计算接口。能和Symbol及宿主语言无缝衔接。
- KVStore: parameter server模式的分布式数据操作接口。使用两级数据同步，机器内同步和机器间同步。支持多种同步模式。
- 内存：使用inplace和co-share的方式减少内存占用。inplace方式统计数据还需要被多少其他单元使用，当不被使用时释放。co-share将不需要并行的数据单元共享同一块内存。
- 其他：提供工具将数据压缩成一个特定格式的包，便于数据预取和多线程操作。

Torch

main site：http://www.torch.ch
github：https://github.com/torch/torch7
torch7文档：http://torch7.readthedocs.io/en/latest/index.html
Cheat sheet：https://github.com/torch/torch7/wiki/Cheatsheet
Model Zoo：https://github.com/torch/torch7/wiki/ModelZoo

luaJIT

使用即时编译（Just-in Time）技术，把 Lua 代码编译成本地机器码后交由 CPU 直接执行，比Lua更快
FFI：Lua代码中申明，连接时直接调用C函数

luarocks：包管理（$ luarocks install image $ luarocks list）
Lua: http://www.lua.org
Lua users: http://lua-users.org/wiki/
LuaJIT: http://luajit.org/luajit.html

TREPL: A REPL for Torch
Torch的一个交互式环境包（Read–eval–print_loop）
https://github.com/torch/trepl/blob/master/README.md

```
$ th -- 打开交互式环境
$ Ctrl+L -- 清屏
$ os.exit() -- 退出，也可用两次 Ctrl+C
```

也可以在luajit解释器中动态加载

```
$ luajit
> repl = require 'trepl'
> repl()
```

功能：

- Tab补全，对命令、函数、变量、文件均有效
- 查看历史，查看历史输出（所有：_RESULTS，上一次：_LAST）
- 自动输出（不需要'='），格式化输出，并且给出执行时间
- 查看帮助：? funcname，环境使用帮助：?
- 执行Shell命令：`$ cmd (example: $ ls)`
- Auto-print after eval (can be stopped with ;)

itorch

https://github.com/facebook/iTorch
问题：
1.itorch notebook启动时报：socket.error: [Errno 99] Cannot assign requested address
实际上是ipython启动时的问题（默认监听localhost），修改启动参数
$ itorch notebook --ip=127.0.0.1
或者
修改/etc/hosts，确保127.0.0.1和localhost唯一对应。

Torchnet

抽象化、模块化的Torch，更简单的异步加载数据、多GPU计算
main page：https://github.com/torchnet/torchnet
模块：Datasets, DatasetIterators, Engines, Meters, Logs

主要模块

torch：N维数组（Tensor）及线性运算（openmp+sse）
image, gnuplot,ffmpeg,audio：图片处理，绘图，视频，音频
nn,rnn,word2vec,dpnn(reinforcement)：神经网络层（Layers as DAG），Loss Fucntion，最优化算法
optim：最优化算法
cutorch, cunn：GPU端的torch和nn实现
cudnn: NVIDIA CUDNN的包装模块

添加层

官方参考：http://torch.ch/docs/developer-docs.html
-- forward过程：
[output] updateOutput(input)
-- backward过程：
[gradInput] updateGradInput(input, gradOutput)
accGradParameters(input, gradOutput, scale) -- 没有参数的层不需要实现该函数

多卡和多机

官方没有多机支持，支持多卡。

- 切换GPU：cutorch.setDevice(devID)
- 所有的cuda调用都是异步的，可以使用cutorch.synchronize()同步

基本步骤为

1. load data
2. loop over GPUs (the loop below will be completely anynchronous, so will run parallely)
   2.1. model[gpuX]:forward
   2.2. criterion[gpuX]:forward
   2.3. criterion[gpuX]:backward
   2.4. model[gpuX]:backward
3. cutorch.synchronize()
4. accumulate GPUx'sgradParameters to GPU1's gradParameters
5. do SGD on GPU1
6. copy back GPU1's parameters to GPUx
7. cutorch.synchronize() and print accuracy etc.

Multi-GPU：

https://github.com/torch/cutorch/issues/42
https://github.com/soumith/imagenet-multiGPU.torch
Multi-Host（parameter server）：https://github.com/sixin-zh/mpiT
lua-mpi(FFI)：https://colberg.org/lua-mpi/README.html
lua-mpi(jzrake)：https://github.com/jzrake/lua-mpi
mpi with docker：https://github.com/ambu50/docker-ib-mpi

使用GPU

```
require 'cunn';
require 'cutorch';
net = net:cuda()
criterion = criterion:cuda()
trainset.data = trainset.data:cuda()
trainset.label = trainset.label:cuda()
trainer = nn.StochasticGradient(net, criterion)
trainer:train(trainset)
```

GPU copy

```
cutorch.setDevice(1)
t1 = torch.randn(100):cuda()
cutorch.setDevice(2)
t2 = torch.randn(100):cuda()
-- NvidiaUVA copy
t2:copy(t1)
```

模型示例

```
-- load data
trainset= torch.load('cifar10-train.t7')
testset= torch.load('cifar10-test.t7')

-- init dataset
setmetatable(trainset,
  {__index = function(t, i)
  return {t.data[i], t.label[i]}
  end}
```

```
);
function trainset:size()
  return self.data:size(1)
end

-- define model
require 'nn';
net = nn.Sequential()
net:add(nn.SpatialConvolution(3, 6, 5, 5))
net:add(nn.ReLU())
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.SpatialConvolution(6, 16, 5, 5))
net:add(nn.ReLU())
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.View(16*5*5))
net:add(nn.Linear(16*5*5, 120))
net:add(nn.ReLU())
net:add(nn.Linear(120, 84))
net:add(nn.ReLU())
net:add(nn.Linear(84, 10))
net:add(nn.LogSoftMax())

-- define loss function
criterion = nn.ClassNLLCriterion()

-- define trainer
trainer = nn.StochasticGradient(net, criterion)
trainer.learningRate = 0.001
trainer.maxIteration = 5

-- training
trainer:train(trainset)

-- predict
predicted = net:forward(testset.data[100])
```

项目

动漫图片无损放大：https://github.com/nagadomi/waifu2x
照片油画风格：https://github.com/jcjohnson/neural-style
图片内容解读：https://github.com/karpathy/neuraltalk2
自动生成文本风格：https://github.com/karpathy/char-rnn
torch-rnn：高效的RNN和LSTM库，用它实现的char-rnn模型比原作快1.9x，内存节省7x：https://github.com/jcjohnson/torch-rnn
用Anaconda安装Torch：https://github.com/alexbw/conda-lua-recipes

数据集

MNIST

数据集大小：~12MB
下载地址：http://yann.lecun.com/exdb/mnist/index.html

MNIST是一个手写数字数据库，它有60000个训练样本集和10000个测试样本集，每个样本图像的宽高为28*28，图像只有一个通道（0（白）-255（黑））。此数据集是以二进制存储的，不能直接以图像格式查看，不过很容易找到将其转换成图像格式的工具。

CIFAR

数据集大小：~170MB
下载地址：http://www.cs.toronto.edu/~kriz/cifar.html

CIFAR-10包含10个类别，50,000个训练图像，彩色图像大小：32×32，10,000个测试图像。CIFAR-100与CIFAR-10类似，包含100个类，每类有600张图片，其中500张用于训练，100张用于测试；这100个类分组成20个超类。图像类别均有明确标注。CIFAR对于图像分类算法测试来说是一个非常

不错的中小规模数据集。

PASCAL VOC

数据集大小：~2GB
下载地址：http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html

PASCAL VOC挑战赛是视觉对象的分类识别和检测的一个基准测试，提供了检测算法和学习性能的标准图像注释数据集和标准的评估系统。PASCAL VOC图片集包括20个目录：人类；动物（鸟、猫、牛、狗、马、羊）；交通工具（飞机、自行车、船、公共汽车、小轿车、摩托车、火车）；室内（瓶子、椅子、餐桌、盆栽植物、沙发、电视）。PASCAL VOC挑战赛在2012年后便不再举办，但其数据集图像质量好，标注完备，非常适合用来测试算法性能。

Imagenet

数据集大小：~1TB（ILSVRC2016比赛全部数据）
下载地址：http://www.image-net.org/about-stats

Imagenet数据集有1400多万幅图片，涵盖2万多个类别；其中有超过百万的图片有明确的类别标注和图像中物体位置的标注，具体信息如下：

1. Total number of non-empty synsets: 21841
2. Total number of images: 14,197,122
3. Number of images with bounding box annotations: 1,034,908
4. Number of synsets with SIFT features: 1000
5. Number of images with SIFT features: 1.2 million

Imagenet数据集是目前深度学习图像领域应用得非常多的一个领域，关于图像分类、定位、检测等研究工作大多基于此数据集展开。Imagenet数据集文档详细，有专门的团队维护，使用非常方便，在计算机视觉领域研究论文中应用非常广，几乎成为了目前深度学习图像领域算法性能检验的“标准”数据集。
与Imagenet数据集对应的有一个享誉全球的“ImageNet国际计算机视觉挑战赛(ILSVRC)”，以往一般是google、MSRA等大公司夺得冠军，今年（2016）ILSVRC2016中国团队包揽全部项目的冠军。
Imagenet数据集是一个非常优秀的数据集，但是标注难免会有错误，几乎每年都会对错误的数据进行修正或是删除，建议下载最新数据集并关注数据集更新。

COCO

数据集大小：~40GB
下载地址：http://mscoco.org/

COCO(Common Objects in Context)是一个新的图像识别、分割和图像语义数据集，它有如下特点：

1. Object segmentation
2. Recognition in Context
3. Multiple objects per image
4. More than 300,000 images
5. More than 2 Million instances
6. 80 object categories
7. 5 captions per image
8. Keypoints on 100,000 people

COCO数据集由微软赞助，其对于图像的标注信息不仅有类别、位置信息，还有对图像的语义文本描述，COCO数据集的开源使得近两三年来图像分割语义理解取得了巨大的进展，也几乎成为了图像语义理解算法性能评价的“标准”数据集。
Google开源的开源了图说生成模型show and tell就是在此数据集上测试的。

Open Image

数据集大小：~1.5GB（不包括图片）
下载地址：https://github.com/openimages/dataset

Open Image是Google推出的一个包含~900万张图像URL的数据集（不包含图片），里面的图片通过标签注释被分为6000多类。该数据集中的标签要比ImageNet（1000类）包含更真实生活的实体存在，它足够让我们从头开始训练深度神经网络。

Youtube-8M

数据集大小：~1.5TB
下载地址：https://research.google.com/youtube8m/

Youtube-8M为谷歌开源的视频数据集，视频来自youtube，共计8百万个视频，总时长50万小时，4800类。为了保证标签视频数据库的稳定性和质量，谷歌只采用浏览量超过1000的公共视频资源。为了让受计算机资源所限的研究者和学生也可以用上这一数据库，谷歌对视频进行了预处理，并提取了帧级别的特征，提取的特征被压缩到可以放到一个硬盘中（小于1.5T）。

数据集资源

1. 深度学习数据集收集网站
   http://deeplearning.net/datasets/**
   收集大量的各深度学习相关的数据集，但并不是所有开源的数据集都能在上面找到相关信息。
2. Tiny Images Dataset
   http://horatio.cs.nyu.edu/mit/tiny/data/index.html
   包含8000万的32×32图像，CIFAR-10和CIFAR-100便是从中挑选的。
3. CoPhIR
   http://cophir.isti.cnr.it/whatis.html
   雅虎发布的超大Flickr数据集，包含1亿多张图片。
4. MirFlickr1M
   http://press.liacs.nl/mirflickr/
   Flickr数据集中挑选出的100万图像集。
5. SBU captioned photo dataset
   http://dsl1.cewit.stonybrook.edu/~vicente/sbucaptions/
   Flickr的一个子集，包含100万的图像集。
6. NUS-WIDE
   http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
   Flickr中的27万的图像集。
7. Large-Scale Image Annotation using Visual Synset(ICCV 2011)
   http://cpl.cc.gatech.edu/projects/VisualSynset/
   机器标注的一个超大规模数据集，包含2亿图像。
8. SUN dataset
   http://people.csail.mit.edu/jxiao/SUN/
   包含13万的图像的数据集。
9. MSRA-MM
   http://research.microsoft.com/en-us/projects/msrammdata/
   包含100万的图像，23000视频；微软亚洲研究院出品，质量应该有保障。

名人

Alex Krizhevsky

主要论文：Imagenet classification with deep convolutional neural networks(2014)
主页：http://www.cs.toronto.edu/~kriz/ (较旧)
Google Scholar：https://scholar.google.com/citations?user=xegzhJcAAAAJ&hl=en