有三 AI 深度学习开源框架
实践指导手册 V1.0

出品单位：有三 AI

作者：言有三

编辑：钦培 言有三等

起草时间：2018 年 9 月—2020 年 6 月

三人行必有 AI

导读

1 开源框架总览

现如今开源生态非常完善，深度学习相关的开源框架众多，光是为人熟知的就有 caffe，tensorflow， pytorch/caffe2， keras， mxnet， paddldpaddle， theano， cntk，deeplearning4j，matconvnet 等。

如何选择最适合你的开源框架是一个问，一个合格的深度学习算法工程师必须熟悉其中所有主流框架，下面是各大开源框架的一个总览。

框架	发布时间	维护组织	底层语言	接口语言	Git star
Caffe	2013/9	BVLC	C++	C++/Python/Matlab	30000+
Tensorflow	2015/9	Google	C++/Python	C++/Python/Java等	140000+
Pytorch	2017/1	Facebook	C/C++/Python	Python	39000+
Mxnet	2015/5	DMLC	C++	C++/Python/Julia/R等	16000+
Keras	2015/3	Google	Python	Python	40000+
Paddlepaddle	2016/8	Baidu	C++/Python	C++/Python	10000+
Cntk	2014/7	Microsoft	C++	C++/Python/C#/.NET/Java	16000+
Matconvnet	2014/2	VLFeat	C/Matlab	Matlab	1300+
Deeplearning4j	2013/9	Skymind	C/C++/Cuda	Java/Scalar等	10000+
Chainer	2015/4	Preferred networks	Python/Cython	Python	5000+
Lasagne/theano	2014/9	Lasagne	C/Python	Python	3600+
Darknet	2013/9	Joseph Redmon	C	C	17000+

在这里我们还有一些框架没有放上来，是因为它们已经升级为大家更喜欢或者使用起来更加简单的版本，比如从 torch->pytorch，从 theano 到 lasagne。另外这些框架都支持 CUDA，因此编程语言这里也没有写上 cuda。

在选择开源框架时，要考虑很多原因，比如开源生态的完善性，比如自己项目的需求，比如自己熟悉的语言。当然，现在已经有很多开源框架之间进行互转的开源工具如 MMDNN 等，也降低了大家迁移框架的学习成本。

除此之外还有 tiny-dnn，ConvNetJS，MarVin，Neon 等等小众，以及 CoreML 等移动端框架，就不再一一介绍。

总的来说对于选择什么样的框架，有三可以给出一些建议：

(1) 不管怎么说，tensorflow/pytorch 你都必须会，这是目前开发者最喜欢，开源项目最丰富的两个框架。

(2) 如果你要进行移动端算法的开发，那么 Caffe 是不能不会的。

(3) 如果你非常熟悉 Matlab，matconvnet 你不应该错过。

(4) 如果你追求高效轻量，那么 darknet 和 mxnet 你不能不熟悉。

(5) 如果你很懒，想写最少的代码完成任务，那么用用 keras 吧。

(6) 如果你是 java 程序员，那么掌握 deeplearning4j 没错的。

其他的框架，也自有它的特点，大家可以自己多去用用。

2 如何学习开源框架

要掌握好一个开源框架，通常需要做到以下几点：

(1) 熟练掌握不同任务数据的准备和使用。

(2) 熟练掌握模型的定义。

(3) 熟练掌握训练过程和结果的可视化。

(4) 熟练掌握训练方法和测试方法。

一个框架，官方都会开放有若干的案例，最常见的案例就是以 MNIST 数据接口+预训练模型的形式，供大家快速获得结果，但是这明显还不够，学习不应该停留在跑通官方的 demo 上，而是要解决实际的问题。

我们要学会从自定义数据读取接口，自定义网络的搭建，模型的训练，模型的可视化，模型的测试与部署等全方位进行掌握，这也是本手册要解决的问题。

目录

【caffe 速成】caffe 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

这一次我们讲讲 Caffe 这个主流的开源框架从训练到测试出结果的全流程。到此，我必须假设大家已经有了深度学习的基础知识并了解卷积网络的工作原理。

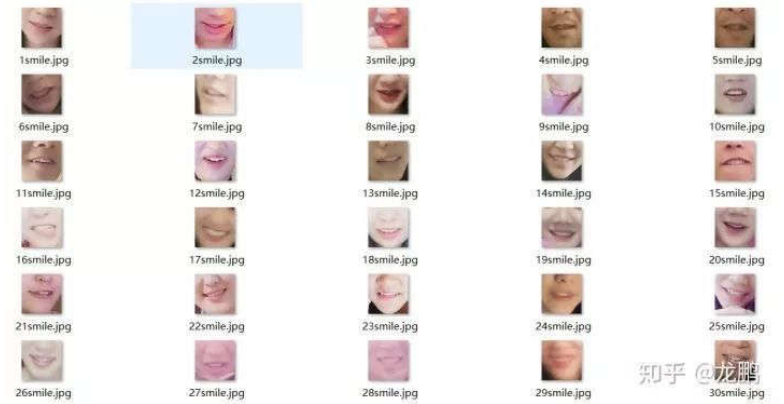
相关的代码、数据都在我们 Git 上，希望大家 Follow 一下这个 Git 项目，后面会持续更新不同框架下的任务。

https://github.com/longpeng2008/yousan.ai

这一篇我们说一个分类任务，给大家准备了 500 张微笑的图片、500 张非微笑的图片，放置在 data 目录下，图片预览如下，已经缩放到 60*60 的大小：

这是非微笑的图片：

这是微笑的图片：



1 Caffe 是什么

Caffe 是以 C++/CUDA 代码为主，最早的深度学习框架之一，比 TensorFlow、Mxnet、Pytorch 等都更早，支持命令行、Python 和 Matlab 接口，单机多卡、多机多卡等都可以很方便的使用，CPU 和 GPU 之间无缝切换。

对于入门级别的任务，如图像分类，Caffe 上手的成本最低，几乎不需要写一行代码就可以开始训练，所以我推荐 Caffe 作为入门学习的框架。

Caffe 相对于 TensorFlow 等使用 pip 一键安装的方式来说，编译安装稍微麻烦一些，但其实依旧很简单，我们以 Ubuntu 16.04 为例子，官网的安装脚本足够用了，它有一些依赖库。

```
sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev
libopencv-dev libhdf5-serial-dev protobuf-compiler
sudo apt-get install --no-install-recommends libboost-all-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev
```

装完之后，到 Git 上 clone 代码，修改 Makefile.config 就可以进行编译安装，如果其中有任何问题，多 Google，还有什么问题，留言吧。当然，对于有 GPU 的读者，还需要安装 cuda 以及 Nvidia 驱动。

2 Caffe 训练

Caffe 完成一个训练，必要准备以下资料：一个是 train.prototxt 作为网络配置文件，另一个是 solver.prototxt 作为优化参数配置文件，再一个是训练文件 list。

另外，在大多数情况下，需要一个预训练模型作为权重的初始化。

(1) 准备网络配置文件

我们准备了一个 3*3 的卷积神经网络，它的 train.prototxt 文件是这样的：

```
name:"mouth"
layer{
name:"data"
type:"ImageData"
top:"data"
top:"clc-label"
image_data_param{
source:"all_shuffle_train.txt"
batch_size:96
```

```
shuffle:true
}
transform_param{
mean_value:104.008
mean_value:116.669
mean_value:122.675
crop_size:48
mirror:true
}
include:{phase:TRAIN}
}
layer{
name:"data"
type:"ImageData"
top:"data"
top:"clc-label"
image_data_param{
source:"all_shuffle_val.txt"
batch_size:30
shuffle:false
 }
transform_param{
mean_value:104.008
mean_value:116.669
mean_value:122.675
crop_size:48
mirror:false
}
include:{phase:TEST}
}
layer{
name:"conv1"
type:"Convolution"
bottom:"data"
top:"conv1"
param{
lr_mult:1
```

```
decay_mult:1
}
param{
lr_mult:2
decay_mult:0
}
convolution_param{
num_output:12
pad:1
kernel_size:3
stride:2
weight_filler{
type:"xavier"
std:0.01
}
bias_filler{
type:"constant"
value:0.2
}
}
}
layer{
name:"relu1"
type:"ReLU"
bottom:"conv1"
top:"conv1"
}
layer{
name:"conv2"
type:"Convolution"
bottom:"conv1"
top:"conv2"
param{
lr_mult:1
decay_mult:1
}
param{
```

```
lr_mult:2
decay_mult:0
}
convolution_param{
num_output:20
kernel_size:3
stride:2
pad:1
weight_filler{
type:"xavier"
std:0.1
}
bias_filler{
type:"constant"
value:0.2
}
}
}
layer{
name:"relu2"
type:"ReLU"
bottom:"conv2"
top:"conv2"
}
layer{
name:"conv3"
type:"Convolution"
bottom:"conv2"
top:"conv3"
param{
lr_mult:1
decay_mult:1
}
param{
lr_mult:2
decay_mult:0
}
```

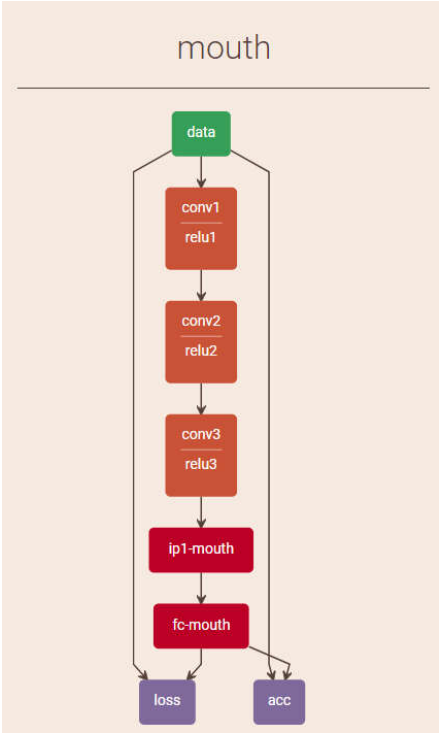
```
convolution_param{
num_output:40
kernel_size:3
stride:2
pad:1
weight_filler{
type:"xavier"
std:0.1
}
bias_filler{
type:"constant"
value:0.2
}
}
}
layer{
name:"relu3"
type:"ReLU"
bottom:"conv3"
top:"conv3"
}
layer{
name:"ip1-mouth"
type:"InnerProduct"
bottom:"conv3"
top:"pool-mouth"
param{
lr_mult:1
decay_mult:1
}
param{
lr_mult:2
decay_mult:0
}
inner_product_param{
num_output:128
weight_filler{
```

```
type:"xavier"
}
bias_filler{
type:"constant"
value:0
}
}
}
layer{
bottom:"pool-mouth"
top:"fc-mouth"
name:"fc-mouth"
type:"InnerProduct"
param{
lr_mult:1
decay_mult:1
}
param{
lr_mult:2
decay_mult:1
}
inner_product_param{
num_output:2
weight_filler{
type:"xavier"
}
bias_filler{
type:"constant"
value:0
}
}
}
layer{
bottom:"fc-mouth"
bottom:"clc-label"
name:"loss"
type:"SoftmaxWithLoss"
```

```
top:"loss"
}
layer{
bottom:"fc-mouth"
bottom:"clc-label"
top:"acc"
name:"acc"
type:"Accuracy"
include{
phase:TRAIN
}
include{
phase:TEST
}
}
```

可以看出，Caffe 的这个网络配置文件，每一个卷积层，都是以 layer{}的形式定义，layer 的 bottom、top 就是它的输入输出，type 就是它的类型，有的是数据层、有的是卷积层、有的是 loss 层。

我们采用 netscope 来可视化一下这个模型。



从上面看很直观的看到，网络的输入层是 data 层，后面接了 3 个卷积层，其中每一个卷积层都后接了一个 relu 层，最后 ip1-mouth、fc-mouth 是全连接层。Loss 和 acc 分别是计算 loss 和 acc 的层。

各层的配置有一些参数，比如 conv1 有卷积核的学习率、卷积核的大小、输出通道数、初始化方法等，这些可以后续详细了解。

(2) 准备训练 list

我们看上面的 data layer，可以到 image_data_param 里面有

```
source: "all_shuffle_train.txt"
```

它是什么呢，就是输入用于训练的 list，它的内容是这样的：

```
../../../../datas/mouth/1/182smile.jpg 1
../../../../datas/mouth/1/435smile.jpg 1
../../../../datas/mouth/0/40neutral.jpg 0
../../../../datas/mouth/1/206smile.jpg 1
../../../../datas/mouth/0/458neutral.jpg 0
../../../../datas/mouth/0/158neutral.jpg 0
../../../../datas/mouth/1/322smile.jpg 1
../../../../datas/mouth/1/83smile.jpg 1
../../../../datas/mouth/0/403neutral.jpg 0
../../../../datas/mouth/1/425smile.jpg 1
../../../../datas/mouth/1/180smile.jpg 1
../../../../datas/mouth/1/233smile.jpg 1
../../../../datas/mouth/1/213smile.jpg 1
../../../../datas/mouth/1/144smile.jpg 1
../../../../datas/mouth/0/327neutral.jpg 0
```

格式就是，图片的名字 + 空格 + label，这就是 Caffe 用于图片分类默认的输入格式。

(3) 准备优化配置文件：

```
net: "./train.prototxt"
test_iter: 100
test_interval: 10
base_lr: 0.00001
momentum: 0.9
type: "Adam"
lr_policy: "fixed"
display: 100
max_iter: 10000
```

```
snapshot: 2000
snapshot_prefix: "./snaps/conv3_finetune"
solver_mode: GPU
```

介绍一下上面的参数。

net 是网络的配置路径。test_interval 是指训练迭代多少次之后，进行一次测试。test_iter 是测试多少个 batch，如果它等于 1，就说明只取一个 batchsize 的数据来做测试，如果 batchsize 太小，那么对于分类任务来说统计出来的指标也不可信，所以最好一次测试，用到所有测试数据。因为，常令 test_iter*test_batchsize=测试集合的大小。

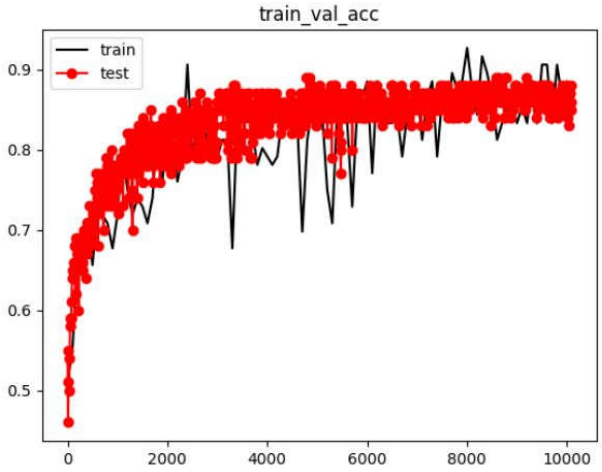
base_lr、momentum、type、lr_policy 是和学习率有关的参数，base_lr 和 lr_policy 决定了学习率大小如何变化。type 是优化的方法，以后再谈。max_iter 是最大的迭代次数，snapshot 是每迭代多少次之后存储迭代结果，snapshot_prefix 为存储结果的目录，caffe 存储的模型后缀是 .caffemodel。solver_mode 可以指定用 GPU 或者 CPU 进行训练。

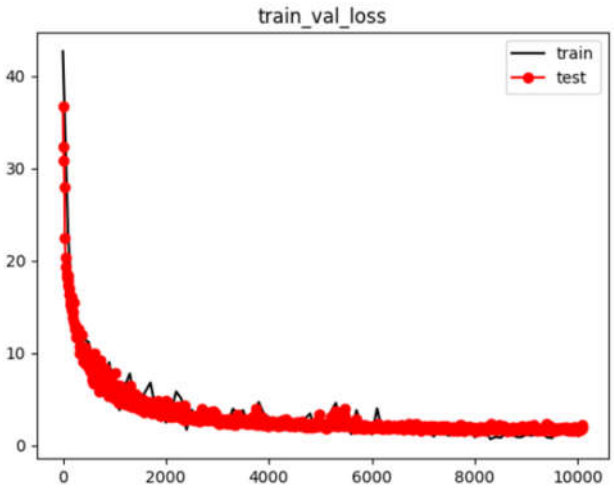
(4) 训练与结果可视化

我们利用 C++ 的接口进行训练，命令如下：

```
SOLVER=./solver.prototxt
WEIGHTS=./init.caffemodel
../../../../libs/Caffe_Long/build/tools/caffe train -solver $SOLVER -weights
$WEIGHTS -gpu 0 2>&1 | tee log.txt
```

其中，caffe train 就是指定训练。我们可以利用脚本可视化一下训练结果，具体参考 git 项目：





3 Caffe 测试

上面我们得到了训练结果，下面开始采用自己的图片进行测试。

train.prototxt 与 test.prototxt 的区别

训练时的网络配置与测试时的网络配置是不同的，测试没有 acc 层，也没有 loss 层，取输出的 softmax 就是分类的结果。同时，输入层的格式也有出入，不需要再输入 label，也不需要指定图片 list，但是要指定输入尺度，我们看一下 test.prototxt 和可视化结果。

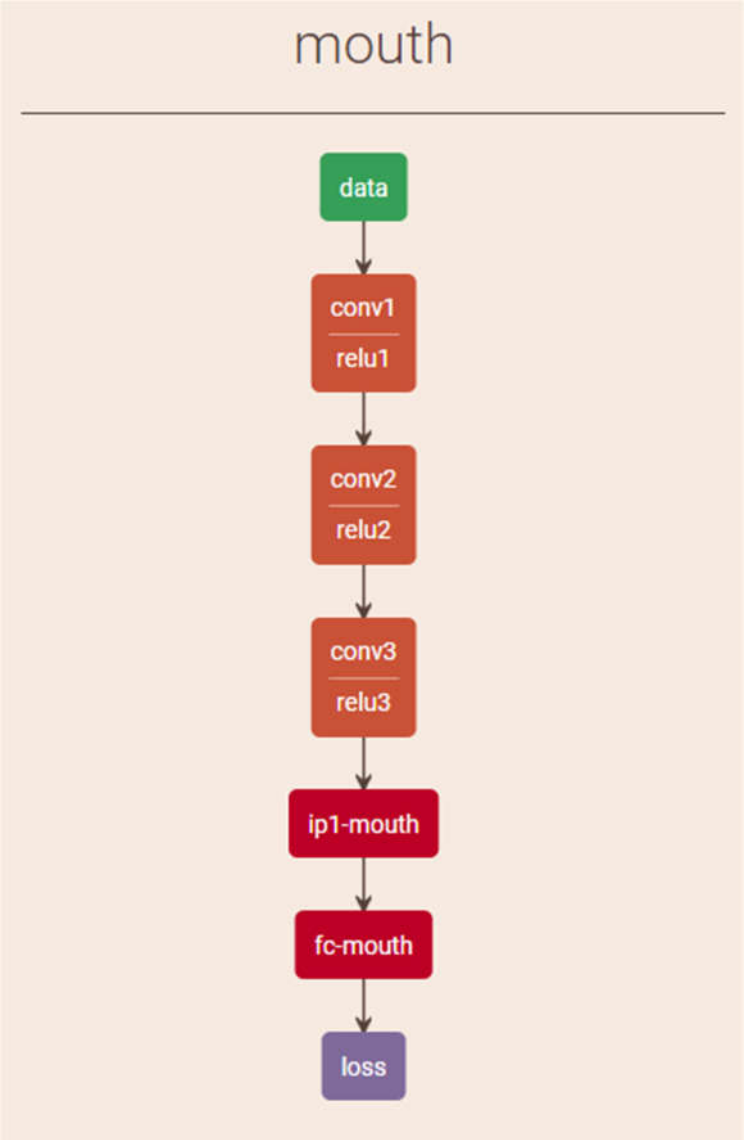
```
name:"mouth"
layer{
name:"data"
type:"Input"
top:"data"
 input_param { shape: { dim: 1 dim: 3 dim: 48 dim: 48 } }
}

layer{
name:"conv1"
type:"Convolution"
bottom:"data"
top:"conv1"
param{
lr_mult:1
decay_mult:1
}
```

```
param{
lr_mult:2
decay_mult:0
}
convolution_param{
num_output:12
pad:1
kernel_size:3
stride:2
weight_filler{
type:"xavier"
std:0.01
}
bias_filler{
type:"constant"
value:0.2
}
}
}
layer{
name:"relu1"
type:"ReLU"
bottom:"conv1"
top:"conv1"
}
layer{
name:"conv2"
type:"Convolution"
bottom:"conv1"
top:"conv2"
param{
lr_mult:1
decay_mult:1
}
param{
lr_mult:2
decay_mult:0
```

```
}
convolution_param{
num_output:20
kernel_size:3
stride:2
pad:1
weight_filler{
type:"xavier"
std:0.1
}
bias_filler{
type:"constant"
value:0.2
}
}
}
layer{
name:"relu2"
type:"ReLU"
bottom:"conv2"
top:"conv2"
}
layer{
name:"conv3"
type:"Convolution"
bottom:"conv2"
top:"conv3"
param{
lr_mult:1
decay_mult:1
}
param{
lr_mult:2
decay_mult:0
}
convolution_param{
num_output:40
```

```
kernel_size:3
stride:2
pad:1
weight_filler{
type:"xavier"
std:0.1
}
bias_filler{
type:"constant"
value:0.2
}
}
}
layer{
name:"relu3"
type:"ReLU"
bottom:"conv3"
top:"conv3"
}
layer{
name:"ip1-mouth"
type:"InnerProduct"
bottom:"conv3"
top:"pool-mouth"
param{
lr_mult:1
decay_mult:1
}
param{
lr_mult:2
decay_mult:0
}
inner_product_param{
num_output:128
weight_filler{
type:"xavier"
}
```

```
bias_filler{
type:"constant"
value:0
}
}
}
layer{
bottom:"pool-mouth"
top:"fc-mouth"
name:"fc-mouth"
type:"InnerProduct"
param{
lr_mult:1
decay_mult:1
}
param{
lr_mult:2
decay_mult:1
}
inner_product_param{
num_output:2
weight_filler{
type:"xavier"
}
bias_filler{
type:"constant"
value:0
}
}
}
layer{
bottom:"fc-mouth"
name:"loss"
type:"Softmax"
top:"prob"
}
```



使用 Python 进行测试

由于 Python 目前广泛使用，下面使用 Python 来进行测试，它要做的就是导入模型、导入图片、输出结果。

下面是所有的代码，我们详细解释下：

```
---代码段 1，这一段，我导入一些基本库，同时导入 caffe 的路径---
#_*_ coding:utf8
import sys
sys.path.insert(0, '../../../../../libs/Caffe_Long/python/')
import caffe
import os,shutil
```

```
import numpy as np
from PIL import Image as PILImage
from PIL import ImageMath
import matplotlib.pyplot as plt
import time
import cv2

---代码段 2，这一段，我们添加一个参数解释器，方便参数管理---
debug=True
import argparse
def parse_args():
  parser = argparse.ArgumentParser(description='test resnet model for
portrait segmentation')
  parser.add_argument('--model', dest='model_proto', help='the model',
default='test.prototxt', type=str)
  parser.add_argument('--weights', dest='model_weight', help='the
weights', default='./test.caffemodel', type=str)
  parser.add_argument('--testsize', dest='testsize', help='inference size',
default=60,type=int)
  parser.add_argument('--src', dest='img_folder', help='the src image
folder', type=str, default='./')
  parser.add_argument('--gt', dest='gt', help='the gt', type=int,
default=0)
  args = parser.parse_args()
  return args

def start_test(model_proto,model_weight,img_folder,testsize):
---代码段 3，这一段，我们就完成了网络的初始化---
  caffe.set_device(0)
  #caffe.set_mode_cpu()
  net = caffe.Net(model_proto, model_weight, caffe.TEST)
  imgs = os.listdir(img_folder)

  pos = 0
  neg = 0

  for imgname in imgs:
```

---代码段 4，这一段，是读取图片并进行预处理，还记得我们之前的训练，是采用 BGR 的输入格式，减去了图像均值吧，同时，输入网络的图像，也需要 resize 到相应尺度。预处理是通过 caffe 的类，transformer 来完成，set_mean 完成均值，set_transpose 完成维度的替换，因为 caffe blob 的格式是 batch、channel、height、width，而 numpy 图像的维度是 height、width、channel 的顺序---

```
    imgtype = imgname.split('.')[-1]
    imgid = imgname.split('.')[0]
    if imgtype != 'png' and imgtype != 'jpg' and imgtype != 'JPG' and
imgtype != 'jpeg' and imgtype != 'tif' and imgtype != 'bmp':
        print imgtype,"error"
        continue
    imgpath = os.path.join(img_folder,imgname)

    img = cv2.imread(imgpath)
    if img is None:
        print "---------img is empty---------",imgpath
        continue

    img = cv2.resize(img,(testsize,testsize))

    transformer = caffe.io.Transformer({'data':
net.blobs['data'].data.shape})
    transformer.set_mean('data', np.array([104.008,116.669,122.675]))
    transformer.set_transpose('data', (2,0,1))
```

---代码段 5，这一段，就得到了输出结果了，并做一些可视化显示---

```
    out = net.forward_all(data=np.asarray([transformer.preprocess('data',
img)]))

    result = out['prob'][0]
    print "---------result prob---------",result,"-------result size--------
",result.shape
    probneutral = result[0]
    print "prob neutral",probneutral
```

```
    probsmile = result[1]
    print "prob smile",probsmile
    problabel = -1
    probstr = 'none'
    if probneutral > probsmile:
        probstr = "neutral:"+str(probneutral)
        pos = pos + 1
    else:
        probstr = "smile:"+str(probsmile)
        neg = neg + 1

    if debug:
      showimg = cv2.resize(img,(256,256))

cv2.putText(showimg,probstr,(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,2
55),1)
        cv2.imshow("test",showimg)
        k = cv2.waitKey(0)
        if k == ord('q'):
            break

  print "pos=",pos
  print "neg=",neg

if __name__ == '__main__':
   args = parse_args()

start_test(args.model_proto,args.model_weight,args.img_folder,args.testsi
ze)
```

经过前面的介绍，我们已经学会了 Caffe 的基本使用，但是我们不能停留于此。Caffe 是一个非常优秀的开源框架，有必要去细读它的源代码。

至于怎么读 Caffe 的代码，建议阅读我写的 Caffe 代码阅读系列内容。

4 总结

虽然现在很多人没有从 Caffe 开始学，但是希望提升自己 C++ 水平和更深刻理解深度学习中的一些源码的，建议从 Caffe 开始学起。

【tensorflow 速成】Tensorflow 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

上一篇介绍了 Caffe ，这篇将介绍 TensorFlow。

1 什么是 TensorFlow

TensorFlow 是 Google brain 推出的开源机器学习库，与 Caffe 一样，主要用作深度学习相关的任务。

与 Caffe 相比 TensorFlow 的安装简单很多，一条 pip 命令就可以解决，新手也不会误入各种坑。

TensorFlow = Tensor + Flow

Tensor 就是张量，代表 N 维数组，与 Caffe 中的 blob 是类似的；Flow 即流，代表基于数据流图的计算。神经网络的运算过程，就是数据从一层流动到下一层，在 Caffe 的每一个中间 layer 参数中，都有 bottom 和 top，这就是一个分析和处理的过程。TensorFlow 更直接强调了这个过程。

TensorFlow 最大的特点是计算图，即先定义好图，然后进行运算，所以所有的 TensorFlow 代码，都包含两部分：

（1）创建计算图，表示计算的数据流。它做了什么呢？实际上就是定义好了一些操作，你可以将它看做是 Caffe 中的 prototxt 的定义过程。

（2）运行会话，执行图中的运算，可以看作是 Caffe 中的训练过程。只是 TensorFlow 的会话比 Caffe 灵活很多，由于是 Python 接口，取中间结果分析，Debug 等方便很多。

2 TensorFlow 训练

咱们这是实战速成，没有这么多时间去把所有事情细节都说清楚，而是抓住主要脉络。有了 TensorFlow 这个工具后，我们接下来的任务就是开始训练模型。训练模型，包括**数据准备、模型定义、结果保存与分析。**

2.1 数据准备

上一节我们说过 Caffe 中的数据准备，只需要准备一个 list 文件，其中每一行存储 image、labelid 就可以了，那是 Caffe 默认的分类网络的 imagedata 层的输入格式。如果想定义自己的输入格式，可以去新建自定义的 Data Layer，而 Caffe 官方的 data

layer 和 imagedata layer 都非常稳定，几乎没有变过，这是我更欣赏 Caffe 的一个原因。因为输入数据，简单即可。相比之下，TensorFlow 中的数据输入接口就要复杂很多，更新也非常快，我知乎有一篇文章，说过从《从 Caffe 到 TensorFlow 1，IO 操作》，有兴趣的读者可以了解一下。

这里我们不再说 TensorFlow 中有多少种数据 IO 方法，先确定好我们的数据格式，那就是跟 Caffe 一样，准备好一个 list，它的格式一样是 image、labelid，然后再看如何将数据读入 TensorFlow 进行训练。

我们定义一个类，叫 imagedata，模仿 Caffe 中的使用方式。代码如下，源代码可移步 Git。

```
import tensorflow as tf
from tensorflow.contrib.data import Dataset
from tensorflow.python.framework import dtypes
from tensorflow.python.framework.ops import convert_to_tensor
import numpy as np
class ImageData:
  def read_txt_file(self):
    self.img_paths = []
    self.labels = []
    for line in open(self.txt_file, 'r'):
      items = line.split(' ')
      self.img_paths.append(items[0])
      self.labels.append(int(items[1]))
  def __init__(self, txt_file, batch_size, num_classes,
               image_size,buffer_scale=100):
    self.image_size = image_size
    self.batch_size = batch_size
    self.txt_file = txt_file ##txt list file,stored as: imagename id
    self.num_classes = num_classes
    buffer_size = batch_size * buffer_scale

    # 读取图片
    self.read_txt_file()
    self.dataset_size = len(self.labels)
    print "num of train datas=",self.dataset_size
    # 转换成 Tensor
```

```
    self.img_paths = convert_to_tensor(self.img_paths,
dtype=dtypes.string)
    self.labels = convert_to_tensor(self.labels, dtype=dtypes.int32)

    # 创建数据集
    data = Dataset.from_tensor_slices((self.img_paths, self.labels))
    print "data type=",type(data)
    data = data.map(self.parse_function)
    data = data.repeat(1000)
    data = data.shuffle(buffer_size=buffer_size)

    # 设置 self data Batch
    self.data = data.batch(batch_size)
    print "self.data type=",type(self.data)

  def augment_dataset(self,image,size):
    distorted_image = tf.image.random_brightness(image, max_delta=63)
    distorted_image =
tf.image.random_contrast(distorted_image,lower=0.2, upper=1.8)
    # Subtract off the mean and divide by the variance of the pixels.
    float_image = tf.image.per_image_standardization(distorted_image)
    return float_image

  def parse_function(self, filename, label):
    label_ = tf.one_hot(label, self.num_classes)
    img = tf.read_file(filename)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.convert_image_dtype(img, dtype = tf.float32)
    img = tf.random_crop(img,[self.image_size[0],
self.image_size[1],3])
     img = tf.image.random_flip_left_right(img)
     img = self.augment_dataset(img,self.image_size)
     return img, label_
```

下面来分析上面的代码，类是 ImageData，它包含几个函数，__init__构造函数，read_txt_file 数据读取函数，parse_function 数据预处理函数，augment_dataset 数据增强函数。

我们直接看构造函数吧，分为几个步骤：

(1)读取变量，文本 list 文件 txt_file，批处理大小 batch_size，类别数 num_classes，要处理成的图片大小 image_size，一个内存变量 buffer_scale=100。

(2)在获取完这些值之后，就到了 read_txt_file 函数。代码很简单，就是利用 self.img_paths 和 self.labels 存储输入 txt 中的文件列表和对应的 label，这一点和 Caffe 很像了。

(3)然后，就是分别将 img_paths 和 labels 转换为 Tensor，函数是 convert_to_tensor，这是 Tensor 内部的数据结构。

(4)创建 dataset，Dataset.from_tensor_slices，这一步，是为了将 img 和 label 合并到一个数据格式，此后我们将利用它的接口，来循环读取数据做训练。当然，创建好 dataset 之后，我们需要给它赋值才能真正的有数据。data.map 就是数据的预处理，包括读取图片、转换格式、随机旋转等操作，可以在这里做。

data = data.repeat(1000) 是将数据复制 1000 份，这可以满足我们训练 1000 个 epochs。data = data.shuffle(buffer_size=buffer_size)就是数据 shuffle 了，buffer_size 就是在做 shuffle 操作时的控制变量，内存越大，就可以用越大的值。

(5)给 selft.data 赋值，我们每次训练的时候，是取一个 batchsize 的数据，所以 self.data = data.batch(batch_size)，就是从上面创建的 dataset 中，一次取一个 batch 的数据。

到此，数据接口就定义完毕了，接下来在训练代码中看如何使用迭代器进行数据读取就可以了。

关于更多 TensorFlow 的数据读取方法，请移步知乎专栏和公众号。

2.2 模型定义

创建数据接口后，我们开始定义一个网络。

```
def simpleconv3net(x):
  x_shape = tf.shape(x)
  with tf.variable_scope("conv3_net"):
     conv1 = tf.layers.conv2d(x, name="conv1",
filters=12,kernel_size=[3,3], strides=(2,2),
activation=tf.nn.relu,kernel_initializer=tf.contrib.layers.xavier_initializer(),b
ias_initializer=tf.contrib.layers.xavier_initializer())
     bn1 = tf.layers.batch_normalization(conv1, training=True,
name='bn1')
```

```
    conv2 = tf.layers.conv2d(bn1, name="conv2",
filters=24,kernel_size=[3,3], strides=(2,2),
activation=tf.nn.relu,kernel_initializer=tf.contrib.layers.xavier_initializer(),b
ias_initializer=tf.contrib.layers.xavier_initializer())
    bn2 = tf.layers.batch_normalization(conv2, training=True,
name='bn2')
    conv3 = tf.layers.conv2d(bn2, name="conv3",
filters=48,kernel_size=[3,3], strides=(2,2),
activation=tf.nn.relu,kernel_initializer=tf.contrib.layers.xavier_initializer(),b
ias_initializer=tf.contrib.layers.xavier_initializer())
    bn3 = tf.layers.batch_normalization(conv3, training=True,
name='bn3')
    conv3_flat = tf.reshape(bn3, [-1, 5 * 5 * 48])
    dense = tf.layers.dense(inputs=conv3_flat, units=128,
activation=tf.nn.relu,name="dense",kernel_initializer=tf.contrib.layers.xavi
er_initializer())
    logits= tf.layers.dense(inputs=dense, units=2,
activation=tf.nn.relu,name="logits",kernel_initializer=tf.contrib.layers.xavi
er_initializer())
    if debug:
      print "x size=",x.shape
      print "relu_conv1 size=",conv1.shape
      print "relu_conv2 size=",conv2.shape
      print "relu_conv3 size=",conv3.shape
      print "dense size=",dense.shape
      print "logits size=",logits.shape
  return logits
```

上面就是我们定义的网络，是一个简单的 3 层卷积。在 tf.layers 下，有各种网络层，这里就用到了 tf.layers.conv2d，tf.layers.batch_normalization 和 tf.layers.dense，分别是卷积层，BN 层和全连接层。我们以一个卷积层为例：

```
conv1 = tf.layers.conv2d(x, name="conv1", filters=12, kernel_size=[3,3],
strides=(2,2), activation=tf.nn.relu,
kernel_initializer=tf.contrib.layers.xavier_initializer(),
bias_initializer=tf.contrib.layers.xavier_initializer())
```

x 即输入，name 是网络名字，filters 是卷积核数量，kernel_size 即卷积核大小，strides 是卷积 stride，activation 即激活函数，kernel_initializer 和

bias_initializer 分别是初始化方法。可见已经将激活函数整合进了卷积层，更全面的参数，请自查 API。其实网络的定义，还有其他接口，tf.nn、tf.layers、tf.contrib，各自重复，在我看来有些混乱。这里之所以用 tf.layers，就是因为参数丰富，适合从头训练一个模型。

2.3 模型训练

老规矩，我们直接上代码，其实很简单。

```
From dataset import *
from net import simpleconv3net
import sys
import os
import cv2

////-------1 定义一些全局变量-------////
txtfile = sys.argv[1]
batch_size = 64
num_classes = 2
image_size = (48,48)
learning_rate = 0.0001

debug=False

if __name__=="__main__":
////-------2 载入网络结构，定义损失函数，创建计算图-------////
  dataset = ImageData(txtfile,batch_size,num_classes,image_size)
  iterator = dataset.data.make_one_shot_iterator()
  dataset_size = dataset.dataset_size
  batch_images,batch_labels = iterator.get_next()
  Ylogits = simpleconv3net(batch_images)

  print "Ylogits size=",Ylogits.shape

  Y = tf.nn.softmax(Ylogits)
  cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=Ylogits,
labels=batch_labels)
  cross_entropy = tf.reduce_mean(cross_entropy)
```

```
  correct_prediction = tf.equal(tf.argmax(Y, 1), tf.argmax(batch_labels, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
  with tf.control_dependencies(update_ops):
     train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy)

  saver = tf.train.Saver()
  in_steps = 100
  checkpoint_dir = 'checkpoints/'
  if not os.path.exists(checkpoint_dir):
     os.mkdir(checkpoint_dir)
  log_dir = 'logs/'
  if not os.path.exists(log_dir):
     os.mkdir(log_dir)
  summary = tf.summary.FileWriter(logdir=log_dir)
  loss_summary = tf.summary.scalar("loss", cross_entropy)
  acc_summary = tf.summary.scalar("acc", accuracy)
  image_summary = tf.summary.image("image", batch_images)
////-3 执行会话，保存相关变量，还可以添加一些 debug 函数来查看中间结果////
 with tf.Session() as sess:
     init = tf.global_variables_initializer()
     sess.run(init)
     steps = 10000
     for i in range(steps):

_,cross_entropy_,accuracy_,batch_images_,batch_labels_,loss_summary_,acc_sum
mary_,image_summary_ =
sess.run([train_step,cross_entropy,accuracy,batch_images,batch_labels,loss_sum
mary,acc_summary,image_summary])
         if i % in_steps == 0 :
            print i,"iterations,loss=",cross_entropy_,"acc=",accuracy_
            saver.save(sess, checkpoint_dir + 'model.ckpt', global_step=i)
            summary.add_summary(loss_summary_, i)
            summary.add_summary(acc_summary_, i)
            summary.add_summary(image_summary_, i)
            #print "predict=",Ylogits," labels=",batch_labels
```

```
            if debug:
                imagedebug = batch_images_[0].copy()
                imagedebug = np.squeeze(imagedebug)
                print imagedebug,imagedebug.shape
                print np.max(imagedebug)
                imagelabel = batch_labels_[0].copy()
                print np.squeeze(imagelabel)

                imagedebug =
cv2.cvtColor((imagedebug*255).astype(np.uint8),cv2.COLOR_RGB2BGR)
                cv2.namedWindow("debug image",0)
                cv2.imshow("debug image",imagedebug)
                k = cv2.waitKey(0)
                if k == ord('q'):
                    break
```

2.4 可视化

TensorFlow 很方便的一点，就是 Tensorboard 可视化。Tensorboard 的具体原理就不细说了，很简单，就是三步。

第一步，创建日志目录。

```
log_dir = 'logs/'
if not os.path.exists(log_dir):        os.mkdir(log_dir)
```

第二步，创建 summary 操作并分配标签，如我们要记录 loss、acc 和迭代中的图片，则创建了下面的变量：

```
loss_summary = tf.summary.scalar("loss", cross_entropy)acc_summary =
tf.summary.scalar("acc", accuracy)image_summary =
tf.summary.image("image", batch_images)
```

第三步，session 中记录结果，如下面代码：

```
_,cross_entropy_,accuracy_,batch_images_,batch_labels_,loss_summary_,ac
c_summary_,image_summary_ =
sess.run([train_step,cross_entropy,accuracy,batch_images,batch_labels,loss
_summary,acc_summary,image_summary])
```

查看训练过程和最终结果时使用：

```
tensorboard --logdir=logs
```

Loss 和 acc 的曲线图如下：

3 TensorFlow 测试

上面已经训练好了模型，我们接下来的目标，就是要用它来做 inference 了。同样给出代码。

```
import tensorflow as tf
from net import simpleconv3net
import sys
import numpy as np
import cv2
import os

testsize = 48
x = tf.placeholder(tf.float32, [1,testsize,testsize,3])
y = simpleconv3net(x)
y = tf.nn.softmax(y)

lines = open(sys.argv[2]).readlines()
count = 0
acc = 0
posacc = 0
negacc = 0
poscount = 0
negcount = 0
```

```
with tf.Session() as sess:
  init = tf.global_variables_initializer()
  sess.run(init)
  saver = tf.train.Saver()
  saver.restore(sess,sys.argv[1])

  #test one by one, you can change it into batch inputs
  for line in lines:
     imagename,label = line.strip().split(' ')
     img = tf.read_file(imagename)
     img = tf.image.decode_jpeg(img,channels = 3)
     img = tf.image.convert_image_dtype(img,dtype = tf.float32)
     img =
tf.image.resize_images(img,(testsize,testsize),method=tf.image.ResizeMet
hod.NEAREST_NEIGHBOR)
     img = tf.image.per_image_standardization(img)

     imgnumpy = img.eval()
     imgs = np.zeros([1,testsize,testsize,3],dtype=np.float32)
     imgs[0:1,] = imgnumpy

     result = sess.run(y, feed_dict={x:imgs})
     result = np.squeeze(result)
     if result[0] > result[1]:
        predict = 0
     else:
        predict = 1

     count = count + 1
     if str(predict) == '0':
        negcount = negcount + 1
        if str(label) == str(predict):
           negacc = negacc + 1
           acc = acc + 1
     else:
        poscount = poscount + 1
```

```
        if str(label) == str(predict):
            posacc = posacc + 1
            acc = acc + 1

    print result
print "acc = ",float(acc) / float(count)
print "poscount=",poscount
print "posacc = ",float(posacc) / float(poscount)
print "negcount=",negcount
print "negacc = ",float(negacc) / float(negcount)
```

从上面的代码可知，与 Train 时同样，需要定义模型，这个跟 Caffe 在测试时使用的 Deploy 是一样的。

然后，用 restore 函数从 saver 中载入参数，读取图像并准备好网络的格式，sess.run 就可以得到最终的结果了。

4 总结

本篇内容讲解了一个最简单的分类例子，相比大部分已封装好的 mnist 或 cifar 为例的代码来说更实用。我们自己准备了数据集，自己设计了网络并进行了结果可视化，学习了如何使用已经训练好的模型做预测。

【pytorch 速成】Pytorch 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

1 什么是 Pytorch

一句话总结 Pytorch = Python + Torch。

Torch 是纽约大学的一个机器学习开源框架，几年前在学术界非常流行，包括 Lecun 等大佬都在使用。但是由于使用的是一种绝大部分人绝对没有听过的 Lua 语言，导致很多人都被吓退。后来随着 Python 的生态越来越完善，Facebook 人工智能研究院推出了 Pytorch 并开源。Pytorch 不是简单的封装 Torch 并提供 Python 接口，而是对 Tensor 以上的所有代码进行了重构，同 TensorFlow 一样，增加了自动求导。

后来 Caffe2 全部并入 Pytorch，如今已经成为了非常流行的框架。很多最新的研究如风格化、GAN 等大多数采用 Pytorch 源码，这也是我们必须要讲解它的原因。

1.1 特点

（1）动态图计算。TensorFlow 从静态图发展到了动态图机制 Eager Execution，pytorch 则一开始就是动态图机制。动态图机制的好处就是随时随地修改，随处 debug，没有类似编译的过程。

（2）简单。相比 TensorFlow 中 Tensor、Variable、Session 等概念充斥，数据读取接口频繁更新，tf.nn、tf.layers、tf.contrib 各自重复，Pytorch 则是从 Tensor 到 Variable 再到 nn.Module，最新的 Pytorch 已经将 Tensor 和 Variable 合并，这分别就是从数据张量到网络的抽象层次的递进。有人调侃 TensorFlow 的设计是“make it complicated”，那么 Pytorch 的设计就是“keep it simple”。

1.2 重要概念

（1）Tensor/Variable

每一个框架都有基本的数据结构，Caffe 是 blob，TensorFlow 和 Pytorch 都是 Tensor，都是高维数组。Pytorch 中的 Tensor 使用与 Numpy 的数组非常相似，两者可以互转且共享内存。

tensor 包括 cpu 和 gpu 两种类型，如 torch.FloatTensortorch.cuda.FloatTensorvirable，就分别表示 cpu 和 gpu 下的 32 位浮点数。

tensor 包含一些属性。data，即 Tensor 内容；Grad，是与 data 对应的梯度；requires_grad，是否容许进行反向传播的学习，更多的可以去查看 API。

（2）nn.module

抽象好的网络数据结构，可以表示为网络的一层，也可以表示为一个网络结构，这是一个基类。在实际使用过程中，经常会定义自己的网络，并继承 nn.Module。具体的使用，我们看下面的网络定义吧。

（3）torchvision 包，包含了目前流行的数据集，模型结构和常用的图片转换工具

2 Pytorch 训练

安装咱们就不说了，接下来的任务就是开始训练模型。训练模型包括数据准备、模型定义、结果保存与分析。

2.1 数据读取

前面已经介绍了 Caffe 和 TensorFlow 的数据读取，两者的输入都是图片 list，但是读取操作过程差异非常大，Pytorch 与这两个又有很大的差异。这一次，直接利用文件夹作为输入，这是 Pytorch 更加方便的做法。数据读取的完整代码如下：

```
data_dir = '../../../../datas/head/'
  data_transforms = {
     'train': transforms.Compose([
        transforms.RandomSizedCrop(48),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.5,0.5,0.5], [0.5,0.5,0.5])
     ]),
     'val': transforms.Compose([
        transforms.Scale(64),
        transforms.CenterCrop(48),
        transforms.ToTensor(),
        transforms.Normalize([0.5,0.5,0.5], [0.5,0.5,0.5])
     ]),
  }
  image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
data_transforms[x]) for x in ['train', 'val']}
  dataloders = {x: torch.utils.data.DataLoader(image_datasets[x],
batch_size=16, shuffle=True, num_workers=4) for x in ['train', 'val']}
```

下面一个一个解释，完整代码请移步 Git 工程。

（1）datasets.ImageFolder

Pytorch 的 torchvision 模块中提供了一个 dataset 包，它包含了一些基本的数据集如 mnist、coco、imagenet 和一个通用的数据加载器 ImageFolder。

它会以这样的形式组织数据，具体的请到 Git 工程中查看。

```
root/left/1.png
root/left/2.png
root/left/3.png

root/right/1.png
root/right/2.png
root/right/3.png
```

imagefolder 有 3 个成员变量。

self.classes：用一个 list 保存类名，就是文件夹的名字。

self.class_to_idx：类名对应的索引，可以理解为 0、1、2、3 等。

self.imgs：保存（imgpath，class），是图片和类别的数组。

不同文件夹下的图，会被当作不同的类，天生就用于图像分类任务。

（2）Transforms

这一点跟 Caffe 非常类似，就是定义了一系列数据集的预处理和增强操作。到此，数据接口就定义完毕了，接下来在训练代码中看如何使用迭代器进行数据读取就可以了，包括 scale、减均值等。

（3）torch.utils.data.DataLoader

这就是创建了一个 batch，生成真正网络的输入。关于更多 Pytorch 的数据读取方法，请自行学习。

2.2 模型定义

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
class simpleconv3(nn.Module):`
def __init__(self):
  super(simpleconv3,self).__init__()
  self.conv1 = nn.Conv2d(3, 12, 3, 2)
  self.bn1 = nn.BatchNorm2d(12)
```

```
 self.conv2 = nn.Conv2d(12, 24, 3, 2)
 self.bn2 = nn.BatchNorm2d(24)
 self.conv3 = nn.Conv2d(24, 48, 3, 2)
 self.bn3 = nn.BatchNorm2d(48)
 self.fc1 = nn.Linear(48 * 5 * 5 , 1200)
 self.fc2 = nn.Linear(1200 , 128)
 self.fc3 = nn.Linear(128 , 2)
def forward(self , x):
 x = F.relu(self.bn1(self.conv1(x)))
 x = F.relu(self.bn3(self.conv3(x)))
 x = x.view(-1 , 48 * 5 * 5)
 x = F.relu(self.fc1(x))
 x = F.relu(self.fc2(x))
 x = self.fc3(x)
 return x
```

我们的例子都是采用一个简单的 3 层卷积 + 2 层全连接层的网络结构。根据上面的网络结构的定义，需要做以下事情。

（1）simpleconv3(nn.Module)

继承 nn.Module，前面已经说过，Pytorch 的网络层是包含在 nn.Module 里，所以所有的网络定义，都需要继承该网络层，并实现 super 方法，如下:

```
super(simpleconv3,self).__init__()
```

这个就当作一个标准执行就可以了。

（2）网络结构的定义都在 nn 包里，举例说明:

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1,
padding=0, dilation=1, groups=1, bias=True)
```

完整的接口如上，定义的第一个卷积层如下：

```
nn.Conv2d(3, 12, 3, 2)
```

即输入通道为 3，输出通道为 12，卷积核大小为 3，stride=2，其他的层就不一一介绍了，大家可以自己去看 nn 的 API。

（3）forward

backward 方法不需要自己实现，但是 forward 函数是必须要自己实现的，从上面可以看出，forward 函数也是非常简单，串接各个网络层就可以了。

对比 Caffe 和 TensorFlow 可以看出，Pytorch 的网络定义更加简单，初始化方法都没有显示出现，因为 Pytorch 已经提供了默认初始化。

如果我们想实现自己的初始化，可以这么做：

```
init.xavier_uniform(self.conv1.weight)init.constant(self.conv1.bias, 0.1)
```

它会对 conv1 的权重和偏置进行初始化。如果要对所有 conv 层使用 xavier 初始化呢？可以定义一个函数：

```
def weights_init(m):
  if isinstance(m, nn.Conv2d):
    xavier(m.weight.data)
    xavier(m.bias.data)
  net = Net()
  net.apply(weights_init)
```

3 模型训练

网络定义和数据加载都定义好之后，就可以进行训练了，老规矩先上代码：

```
def train_model(model, criterion, optimizer, scheduler,  num_epochs=25):
  for epoch in range(num_epochs):
   print('Epoch {}/{}'.format(epoch, num_epochs - 1))
   for phase in ['train', 'val']:
    if phase == 'train':
      scheduler.step()
      model.train(True)
    else:
      model.train(False)
      running_loss = 0.0
running_corrects = 0.0
    for data in dataloders[phase]:
      inputs, labels = data
      if use_gpu:
        inputs = Variable(inputs.cuda())
        labels = Variable(labels.cuda())
      else:
        inputs, labels = Variable(inputs), Variable(labels)

       optimizer.zero_grad()
       outputs = model(inputs)
       _, preds = torch.max(outputs.data, 1)
```

```
        loss = criterion(outputs, labels)
        if phase == 'train':
         loss.backward()
         optimizer.step()

        running_loss += loss.data.item()
        running_corrects += torch.sum(preds == labels).item()

    epoch_loss = running_loss / dataset_sizes[phase]
      epoch_acc = running_corrects / dataset_sizes[phase]

      if phase == 'train':
        writer.add_scalar('data/trainloss', epoch_loss, epoch)
        writer.add_scalar('data/trainacc', epoch_acc, epoch)
      else:
       writer.add_scalar('data/valloss', epoch_loss, epoch)
       writer.add_scalar('data/valacc', epoch_acc, epoch)

      print('{} Loss: {:.4f} Acc: {:.4f}'.format(
              phase, epoch_loss, epoch_acc))
  writer.export_scalars_to_json("./all_scalars.json")
  writer.close()
  return model
```

分析一下上面的代码，外层循环是 epoches，然后利用 for data in dataloders[phase] 循环取一个 epoch 的数据，并塞入 variable，送入 model。需要注意的是，每一次 forward 要将梯度清零，即 optimizer.zero_grad()，因为梯度会记录前一次的状态，然后计算 loss 进行反向传播。

```
loss.backward()
optimizer.step()
```

下面可以分别得到预测结果和 loss，每一次 epoch 完成计算。

```
epoch_loss = running_loss / dataset_sizes[phase]
epoch_acc = running_corrects / dataset_sizes[phase]
_, preds = torch.max(outputs.data, 1)
loss = criterion(outputs, labels)
```

可视化是非常重要的，鉴于 TensorFlow 的可视化非常方便，我们选择了一个开源工具包，tensorboardx，安装方法为 pip install tensorboardx，使用非常简单。

第一步，引入包定义创建：

```
from tensorboardX import SummaryWriter
writer = SummaryWriter()
```

第二步，记录变量，如 train 阶段的 loss，writer.add_scalar('data/trainloss', epoch_loss, epoch)。

按照以上操作就完成了，完整代码可以看配套的 Git 项目，我们看看训练中的记录。Loss 和 acc 的曲线图如下：

网络的收敛没有 Caffe 和 TensorFlow 好，大家可以自己去调试调试参数了，随便折腾吧。

4 Pytorch 测试

上面已经训练好了模型，接下来的目标就是要用它来做 inference 了，同样给出代码。

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.autograd import Variable
import torchvision
from torchvision import datasets, models, transforms
import time
import os
from PIL import Image
import sys
import torch.nn.functional as F
```

```
from net import simpleconv3
data_transforms =  transforms.Compose([
            transforms.Resize(48),
            transforms.ToTensor(),
            transforms.Normalize([0.5,0.5,0.5], [0.5,0.5,0.5])])

net = simpleconv3()
modelpath = sys.argv[1]
net.load_state_dict(torch.load(modelpath,map_location=lambda
storage,loc: storage))

imagepath = sys.argv[2]
image = Image.open(imagepath)
imgblob = data_transforms(image).unsqueeze(0)
imgblob = Variable(imgblob)
torch.no_grad()
predict = F.softmax(net(imgblob))
print(predict)
```

从上面的代码可知，做了几件事：

定义网络并使用 torch.load 和 load_state_dict 载入模型。

用 PIL 的 Image 包读取图片，这里没有用 OpenCV，因为 Pytorch 默认的图片读取工具就是 PIL 的 Image，它会将图片按照 RGB 的格式，归一化到 0～1 之间。读取图片之后，必须转化为 Tensor 变量。

evaluation 的时候，必须设置 torch.no_grad()，然后就可以调用 softmax 函数得到结果了。

5 总结

本节讲了如何用 Pytorch 完成一个分类任务，并学习了可视化以及使用训练好的模型做测试。

【paddlepaddle 速成】paddlepaddle 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

这一次我们讲讲 paddlepadle 这个百度开源的机器学习框架，一个图像分类任务从训练到测试出结果的全流程。

将涉及到 paddlepaddle 和 visualdl，git 如下：*https://github.com/PaddlePaddle*

相关的代码、数据都在我们 Git 上，希望大家 Follow 一下这个 Git 项目，后面会持续更新不同框架下的任务。

https://github.com/longpeng2008/yousan.ai

1 paddlepaddle 是什么

正所谓 google 有 tensorflow，facebook 有 pytorch，amazon 有 mxnet，作为国内机器学习的先驱，百度也有 PaddlePaddle，其中 Paddle 即 Parallel Distributed Deep Learning(并行分布式深度学习)，整体使用起来与 tensorflow 非常类似。

```
sudo pip install paddlepaddle
```

安装就是一条命令，话不多说上代码。

2 paddlepaddle 训练

训练包括三部分，数据的定义，网络的定义，以及可视化和模型的存储。

2.1 数据定义

定义一个图像分类任务的 dataset 如下：

```
from multiprocessing import cpu_count
import paddle.v2 as paddle

class Dataset:
 def __init__(self,cropsize,resizesize):
       self.cropsize = cropsize
       self.resizesize = resizesize

def train_mapper(self,sample):
```

```
        img, label = sample
        img = paddle.image.load_image(img)
        img = paddle.image.simple_transform(img, self.resizesize,
self.cropsize, True)
        #print "train_mapper",img.shape,label
        return img.flatten().astype('float32'), label

def test_mapper(self,sample):
        img, label = sample
        img = paddle.image.load_image(img)
        img = paddle.image.simple_transform(img, self.resizesize,
self.cropsize, False)
        #print "test_mapper",img.shape,label
        return img.flatten().astype('float32'), label

def train_reader(self,train_list, buffered_size=1024):
    def reader():
        with open(train_list, 'r') as f:
            lines = [line.strip() for line in f.readlines()]
            print "len of train dataset=",len(lines)
            for line in lines:
                img_path, lab = line.strip().split(' ')
                yield img_path, int(lab)

    return paddle.reader.xmap_readers(self.train_mapper, reader,
                                      cpu_count(), buffered_size)

 def test_reader(self,test_list, buffered_size=1024):
    def reader():
        with open(test_list, 'r') as f:
            lines = [line.strip() for line in f.readlines()]
            print "len of val dataset=",len(lines)
            for line in lines:
                img_path, lab = line.strip().split(' ')
                yield img_path, int(lab)

    return paddle.reader.xmap_readers(self.test_mapper, reader,
```

```
                                cpu_count(), buffered_size)
```

从上面代码可以看出:

(1) 使用了 paddle.image.load_image 进行图片的读取，paddle.image.simple_transform 进行了简单的图像变换，这里只有图像 crop 操作，更多的使用可以参考 API。

(2) 使用了 paddle.reader.xmap_readers 进行数据的映射。

2.2 网络定义

```
# coding=utf-8
import paddle.fluid as fluid
def simplenet(input):
  # 定义卷积块
  conv1 = fluid.layers.conv2d(input=input, num_filters=12, stride=2,
padding=1, filter_size=3, act="relu")
  bn1 = fluid.layers.batch_norm(input=conv1)
  conv2 = fluid.layers.conv2d(input=bn1, num_filters=12, stride=2,
padding=1, filter_size=3, act="relu")
  bn2 = fluid.layers.batch_norm(input=conv2)
  conv3 = fluid.layers.conv2d(input=bn2,
num_filters=12,stride=2,padding=1,filter_size=3,act="relu")
  bn3 = fluid.layers.batch_norm(input=conv3)
  fc1 = fluid.layers.fc(input=bn3, size=128, act=None)
  return fc1,conv1
```

与之前的 caffe，pytorch，tensorflow 框架一样，定义了一个 3 层卷积与 2 层全连接的网络。为了能够更好的进行可视化，我们使用了 PaddlePaddle Fluid，Fluid 的设计也是用来让用户像 Pytorch 和 Tensorflow Eager Execution 一样可以执行动态计算而不需要创建图。

2.3 可视化

paddlepaddle 有与之配套使用的可视化框架，即 visualdl。

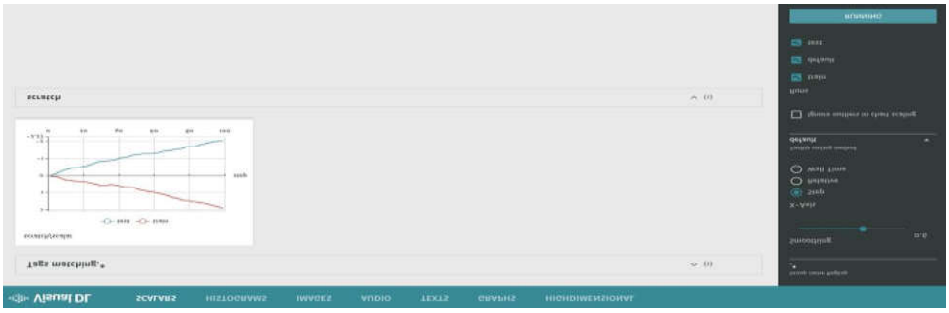
visualdl 是百度数据可视化实验室发布的深度学习可视化平台，它的定位与 tensorboard 很像，可视化内容包含了向量，参数直方图分布，模型结构，图像等功能，以后我们会详细给大家讲述，这次直接在代码中展示如何使用。

安装使用 pip install --upgrade visualdl，使用下面的命令可以查看官方例子：

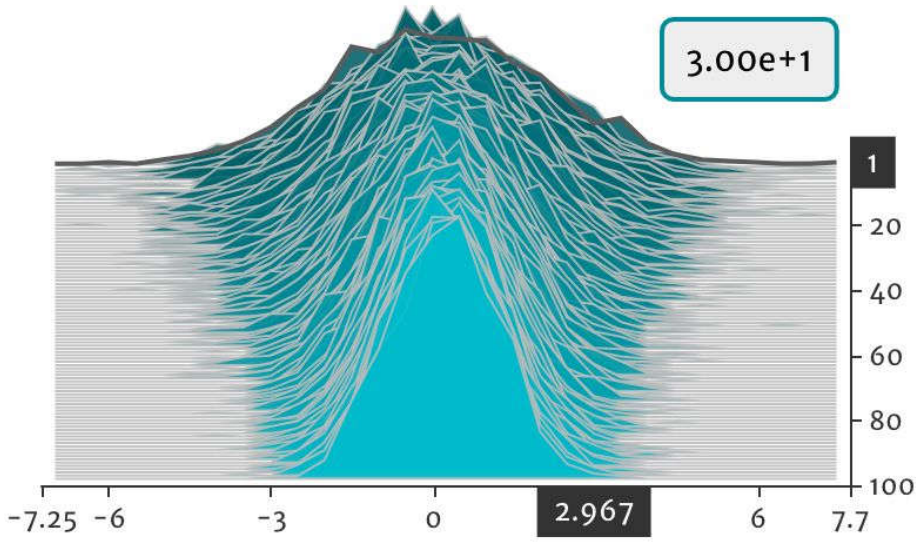
```
vdl_create_scratch_log
```

```
visualDL --logdir ./scratch_log --port 8080
http://127.0.0.1:8080
```

下面是 loss 和直方图的查看



scratch/histogram(train)



在咱们项目中，具体使用方法如下：

```
# 首先定义相关变量
# 创建 VisualDL，并指定 log 存储路径
logdir = "./logs"
logwriter = LogWriter(logdir, sync_cycle=10)

# 创建 loss 的趋势图
with logwriter.mode("train") as writer:
  loss_scalar = writer.scalar("loss")
```

```
# 创建 acc 的趋势图
with logwriter.mode("train") as writer:
  acc_scalar = writer.scalar("acc")

# 定义输出频率
num_samples = 4
# 创建卷积层和输出图像的图形化展示
with logwriter.mode("train") as writer:
  conv_image = writer.image("conv_image", num_samples, 1)
  input_image = writer.image("input_image", num_samples, 1)

# 创建可视化的训练模型结构
with logwriter.mode("train") as writer:
  param1_histgram = writer.histogram("param1", 100)
```

然后在训练过程中进行记录，这是完整的训练代码，红色部分就是记录结果。

```
# coding=utf-8
import numpy as np
import os
import paddle.fluid as fluid
import paddle.fluid.framework as framework
import paddle.v2 as paddle
from paddle.fluid.initializer import NormalInitializer
from paddle.fluid.param_attr import ParamAttr
from visualdl import LogWriter
from dataset import Dataset
from net_fluid import simplenet

# 创建 VisualDL，并指定当前该项目的 VisualDL 的路径
logdir = "./logs"
logwriter = LogWriter(logdir, sync_cycle=10)

# 创建 loss 的趋势图
with logwriter.mode("train") as writer:
  loss_scalar = writer.scalar("loss")

# 创建 acc 的趋势图
```

```
with logwriter.mode("train") as writer:
  acc_scalar = writer.scalar("acc")

# 定义输出频率
num_samples = 4
# 创建卷积层和输出图像的图形化展示
with logwriter.mode("train") as writer:
  conv_image = writer.image("conv_image", num_samples, 1)
  input_image = writer.image("input_image", num_samples, 1)

# 创建可视化的训练模型结构
with logwriter.mode("train") as writer:
  param1_histgram = writer.histogram("param1", 100)

def train(use_cuda, learning_rate, num_passes, BATCH_SIZE=128):
  class_dim = 2
  image_shape = [3, 48, 48]
  image = fluid.layers.data(name='image', shape=image_shape,
dtype='float32')
  label = fluid.layers.data(name='label', shape=[1], dtype='int64')

  net, conv1 = simplenet(image)
  # 获取全连接输出
  predict = fluid.layers.fc(
      input=net,
      size=class_dim,
      act='softmax',
      param_attr=ParamAttr(name="param1",
initializer=NormalInitializer()))

  # 获取损失
  cost = fluid.layers.cross_entropy(input=predict, label=label)
  avg_cost = fluid.layers.mean(x=cost)

  # 计算 batch，从而来求平均的准确率
  batch_size = fluid.layers.create_tensor(dtype='int64')
  print "batchsize=",batch_size
```

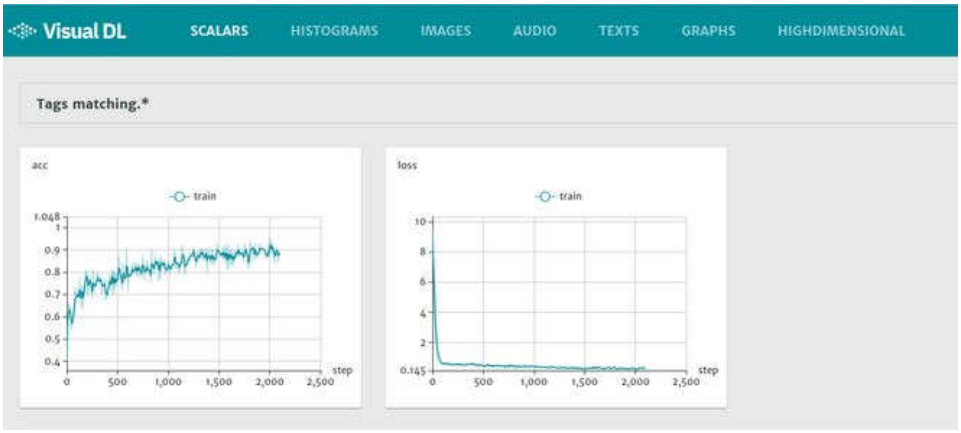
```
  batch_acc = fluid.layers.accuracy(input=predict, label=label,
total=batch_size)

  # 定义优化方法
  optimizer = fluid.optimizer.Momentum(
      learning_rate=learning_rate,
      momentum=0.9,
      regularization=fluid.regularizer.L2Decay(5 * 1e-5))

  opts = optimizer.minimize(avg_cost)

  # 是否使用 GPU
  place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()
  # 创建调试器
  exe = fluid.Executor(place)
  # 初始化调试器
  exe.run(fluid.default_startup_program())
  # 保存结果
  model_save_dir = "./models"

  # 获取训练数据
  resizesize = 60
  cropsize = 48
  mydata = Dataset(cropsize=cropsize,resizesize=resizesize)
  mydatareader = mydata.train_reader(train_list='./all_shuffle_train.txt')
  train_reader =
paddle.batch(reader=paddle.reader.shuffle(reader=mydatareader,buf_size
=50000),batch_size=128)

  # 指定数据和 label 的对应关系
  feeder = fluid.DataFeeder(place=place, feed_list=[image, label])

  step = 0
  sample_num = 0
  start_up_program = framework.default_startup_program()
  param1_var = start_up_program.global_block().var("param1")
```

```
  accuracy = fluid.average.WeightedAverage()
  # 开始训练，使用循环的方式来指定训多少个 Pass
  for pass_id in range(num_passes):
      # 从训练数据中按照一个个 batch 来读取数据
      accuracy.reset()
      for batch_id, data in enumerate(train_reader()):
          loss, conv1_out, param1, acc, weight =
exe.run(fluid.default_main_program(),
                                              feed=feeder.feed(data),
                                              fetch_list=[avg_cost, conv1,
param1_var, batch_acc,batch_size])
          accuracy.add(value=acc, weight=weight)
          pass_acc = accuracy.eval()

          # 重新启动图形化展示组件
          if sample_num == 0:
              input_image.start_sampling()
              conv_image.start_sampling()
          # 获取 taken
          idx1 = input_image.is_sample_taken()
          idx2 = conv_image.is_sample_taken()
          # 保证它们的 taken 是一样的
          assert idx1 == idx2
          idx = idx1
          if idx != -1:
              # 加载输入图像的数据数据
              image_data = data[0][0]
              input_image_data = np.transpose(
                  image_data.reshape(image_shape), axes=[1, 2, 0])
              input_image.set_sample(idx, input_image_data.shape,
                                     input_image_data.flatten())
              # 加载卷积数据
              conv_image_data = conv1_out[0][0]
              conv_image.set_sample(idx, conv_image_data.shape,
                                    conv_image_data.flatten())
              # 完成输出一次
              sample_num += 1
```

```
            if sample_num % num_samples == 0:
                input_image.finish_sampling()
                conv_image.finish_sampling()
                sample_num = 0

        # 加载趋势图的数据
        loss_scalar.add_record(step, loss)
        acc_scalar.add_record(step, acc)
        # 添加模型结构数据
        param1_histgram.add_record(step, param1.flatten())

        # 输出训练日志
        print("loss:" + str(loss) + " acc:" + str(acc) + " pass_acc:" +
str(pass_acc))
        step += 1
        model_path = os.path.join(model_save_dir,str(pass_id))
        if not os.path.exists(model_save_dir):
            os.mkdir(model_save_dir)
        fluid.io.save_inference_model(model_path,['image'],[predict],exe)

if __name__ == '__main__':
  # 开始训练
  train(use_cuda=False, learning_rate=0.005, num_passes=300)
```

2.4 训练结果

看看 acc 和 loss 的曲线，可见已经收敛



3 paddlepaddle 测试

训练的时候使用了 fluid，测试的时候也需要定义调试器，加载训练好的模型，完整的代码如下

```
# encoding:utf-8
import sys
import numpy as np
import paddle.v2 as paddle
from PIL import Image
import os
import cv2
# coding=utf-8
import numpy as np
import paddle.fluid as fluid
import paddle.fluid.framework as framework
import paddle.v2 as paddle
from paddle.fluid.initializer import NormalInitializer
from paddle.fluid.param_attr import ParamAttr
from visualdl import LogWriter
from net_fluid import simplenet

if __name__ == "__main__":
  # 开始预测
  type_size = 2
  testsize = 48

  imagedir = sys.argv[1]
  images = os.listdir(imagedir)

  # 定义调试器
  save_dirname = "./models/299"
  exe = fluid.Executor(fluid.CPUPlace())
  inference_scope = fluid.core.Scope()
  with fluid.scope_guard(inference_scope):
  # 加载模型
```

```
 [inference_program,feed_target_names,fetch_targets] =
fluid.io.load_inference_model(save_dirname,exe)

      predicts = np.zeros((type_size,1))
      for image in images:
         imagepath = os.path.join(imagedir,image)
         img = paddle.image.load_image(imagepath)
         img = paddle.image.simple_transform(img,testsize,testsize,False)
         img = img[np.newaxis,:]

         #print img.shape

         results = np.argsort(-
exe.run(inference_program,feed={feed_target_names[0]:img},
                fetch_list=fetch_targets)[0])
         label = results[0][0]
         predicts[label] += 1

   print predicts
```

由于所有框架的测试流程都差不多，所以就不对每一部分进行解释了，大家可以自行去看代码。

【Keras 速成】Keras 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

这一次我们讲讲 keras 这个简单、流行的深度学习框架，一个图像分类任务从训练到测试出结果的全流程。

相关的代码、数据都在我们 Git 上，希望大家 Follow 一下这个 Git 项目，后面会持续更新不同框架下的任务。

https://github.com/longpeng2008/yousan.ai

1 keras 是什么

Keras 是一个非常流行、简单的深度学习框架，它的设计参考了 torch，用 Python 语言编写，是一个高度模块化的神经网络库，支持 GPU 和 CPU。能够在 TensorFlow，CNTK 或 Theano 之上运行。 Keras 的特点是能够快速实现模型的搭建， 简单方便地让你实现从想法到实验验证的转化，这都是高效地进行科学研究的关键。

2 Keras 安装配置

Keras 的安装非常简单，但是需要先安装一个后端框架作为支撑，TensorFlow， CNTK，Theano 都可以，但是官网上强烈建议使用 TensorFlow 作为 Keras 的后端进行使用。本例以 TensorFlow 1.4.0 版本作为 Keras 的后端进行测试。

```
sudo pip install tensorflow==1.4.0
sudo pip install keras==2.1.4
```

通过上面两条命令就可以完成 TensorFlow 和 Keras 的安装，此处需要注意的一点是 Keras 的版本和 TensorFlow 的版本要对应，否则会出现意外的错误。具体版本对应关系可在网上进行查询。

3 Keras 自定义数据

3.1 MNIST 实例

MNIST 手写字符分类被认为是深度学习框架里的“Hello Word！”，下面简单介绍一下 MNIST 数据集案例的测试。Keras 的官方 github 的 example 目录下提供了几个 MNIST 案例的代码，下载 mnist_mlp.py，mnist_cnn.py 文件，本地运行即可，其他文件读者也可以自行测试。

3.2 数据定义

前面我们介绍了 MNIST 数据集实例，很多读者在学习深度学习框架的时候都卡在了这一步，运行完 MNIST 实例之后无从下手，很大原因可能是因为不知道怎么处理自己的数据集，这一节我们通过一个简单的图像二分类案例，介绍如何实现一个自定义的数据集。

数据处理有几种方式，一种是像 MNIST、CIFAR 数据集，这些数据集的特点是已经为用户打包封装好了数据。用户只要 load_data 即可实现数据导入。其实就是事先把数据进行解析，然后保存到.pkl 或者.h5 等文件中，然后在训练模型的时候直接导入，输入到网络中；另一种是直接从本地读取文件，解析成网络需要的格式，输入网络进行训练。但是实际情况是，为了某一个项目我们不可能总是找到相应的打包好的数据集供使用，这时候自己建立一个 dataset 就十分重要。

Keras 提供了一个图像数据的数据增强文件，调用这个文件我们可以实现网络数据加载的功能。

此处采用 keras 的 processing 模块里的 ImageDataGenerator 类定义一个图像分类任务的 dataset 生成器：

```
train_data_dir = '../../../../datas/head/train/'
validation_data_dir = '../../../../datas/head/val'
# augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1. / 255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
# augmentation configuration use for testing only rescaling
val_datagen = ImageDataGenerator(rescale=1. / 255)
train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(48, 48),
        batch_size=16)
val_generator = val_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(48, 48),
        batch_size=16)
```

下面简单地介绍一下上面的代码，完整代码请移步 Git 工程。

Keras 的 processing 模块中提供了一个能够实时进行数据增强的图像生成类 ImagGenerator，该类下面有一个函数 flow_from_directory，顾名思义该函数就是从文件夹中获取图像数据。关于 ImageGenerator 更多的使用可以参考官方源码。数据集结构组织如下：

```
datas/train/left/*.jpg
datas/train/right/*.jpg
datas/val/left/*.jpg
datas/val/right/*.jpg
```

此处还需要注意的一点是，我们现在进行的是简单的图像分类任务训练，假如要完成语义分割，目标检测等任务，则需要自定义一个类（继承 ImageDataGenerator），具体实现可以查询相关代码进行参考。

4 Keras 网络搭建

Keras 网络模型搭建有两种形式，Sequential 顺序模型和使用函数式 API 的 Model 类模型。本教程的例子采用一个简单的三层卷积，以及两层全连接和一个分类层组成的网络模型。由于函数式 API 更灵活方便，因此下面采用函数式方法搭建模型，模型定义如下：

4.1 函数式 API

```
def simpleconv3(input_shape=(48, 48, 3), classes=2):
    img_input = Input(shape=input_shape)
    bn_axis = 3
    x = Conv2D(12, (3, 3), strides=(2, 2), padding='same',
name='conv1')(img_input)
    x = BatchNormalization(axis=bn_axis, name='bn_conv1')(x)
    x = Activation('relu')(x)
    x = Conv2D(24, (3, 3), strides=(2, 2), padding='same', name='conv2')(x)
    x = BatchNormalization(axis=bn_axis, name='bn_conv2')(x)
    x = Activation('relu')(x)
    x = Conv2D(48, (3, 3), strides=(2, 2), padding='same', name='conv3')(x)
    x = BatchNormalization(axis=bn_axis, name='bn_conv3')(x)
    x = Activation('relu')(x)
    x = Flatten()(x)
    x = Dense(1200, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    x = Dense(classes, activation='softmax')(x)
```

```
    model = Model(img_input, x)
    return model
x = Conv2D(12, (3, 3), strides=(2, 2), padding='same',
name='conv1')(img_input)
```

即输出是 12 通道，卷积核大小 3*3，步长为 2，padding=’same’表示边缘补零

```
x = BatchNormalization(axis=bn_axis, name='bn_conv1')(x)
```

axis 表示需要归一化的坐标轴，bn_axis=3，由于采用 TensorFlow 作为后端，因此这句代码表示在通道数坐标轴进行归一化。

x = Flatten()(x) 表示将卷积特征图进行拉伸，以便和全连接层 Dense()进行连接。

```
x = Dense(1200, activation='relu')(x)
```

Dense()实现全连接层的功能，1200 是输出维度，‘relu’表示激活函数，使用其他函数可以自行修改。

最后一层采用‘softmax’激活函数实现分类功能。

最终返回 Model，包含网络的输入和输出。

4.2 模型编译

网络搭建完成，在网络训练前需要进行编译，包括学习方法、损失函数、评估标准等，这些参数分别可以从 optimizer、loss、metric 模块中导入。具体代码如下：

```
from keras.optimizers import SGD
from keras.losses import binary_crossentropy
from keras.metrics import binary_accuracy
from keras.callbacks import TensorBoard

tensorboard = TensorBoard(log_dir=('./logs'))
callbacks = []
callbacks.append(tensorboard)
loss = binary_crossentropy
metrics = [binary_accuracy]
optimizer = SGD(lr=0.001, decay=1e-6, momentum=0.9)
```

其中 callbacks 模块包含了 TensorBoard， ModelCheckpoint，LearningRateScheduler 等功能，分别可以用来可视化模型，设置模型检查点，以及设置学习率策略。

5 模型训练、测试

5.1 模型训练

Keras 模型训练过程非常简单，只需一行代码，设置几个参数即可，具体代码如下：

```
history = model.fit_generator(
        train_generator,
        steps_per_epoch=num_train_samples // batch_size,
        epochs=epochs,
        callbacks=callbacks,
        validation_data=val_generator,
        validation_steps=num_val_samples // batch_size)
```

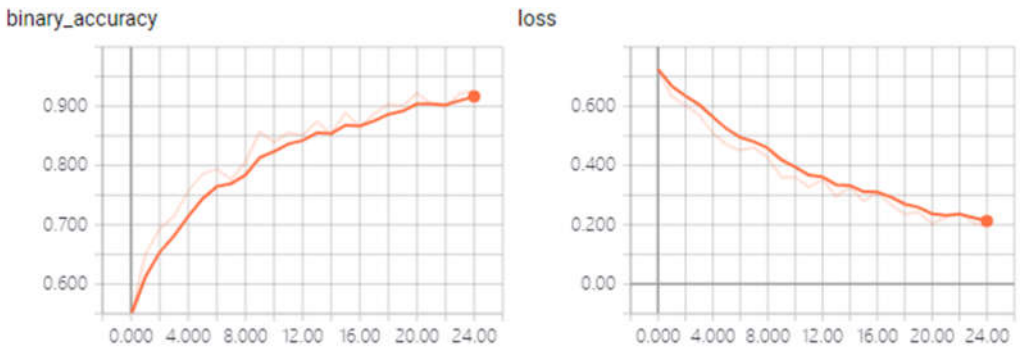
首先指定数据生成器，train_generator， 前面介绍过；steps_per_epoch 是每次 epoch 循环的次数，通过训练样本数除以 batch_size 得到；epochs 是整个数据集重复多少次训练。

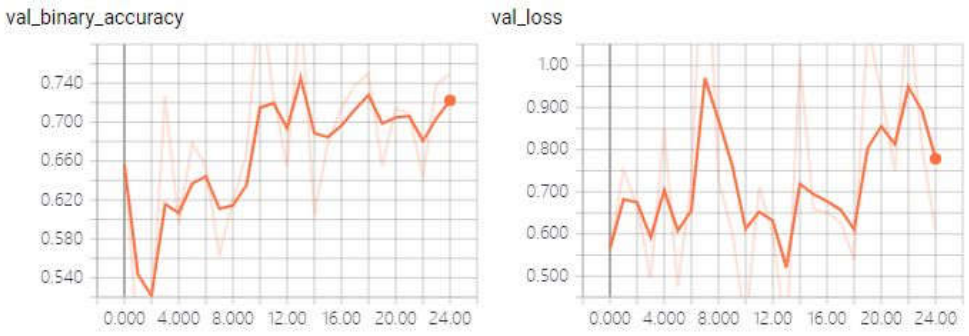
Keras 是高度封装的，在模型训练过程中，看不到网络的预测结果和网络的反向传播过程，只需定义好损失函数，事实上，网络定义中的模型输出会包含网络的输入和输出。

5.2 训练过程可视化

keras 可以采用 tensorboard 实现训练过程的可视化。执行完下面的命令就可以在浏览器访问 http://127.0.0.1:6006 查看效果。

```
tensorboard --logdir 日志文件路径（默认路径='./logs''）
```





上面是分别是训练和测试过程的 loss 和 accuracy。

5.3 模型测试

```
model = simpleconv3()
model.load_weights(model_path, by_name=True)
image_path = '../../../../datas/head/train/0/1left.jpg'
img = Image.open(image_path)
img = img_to_array(img)
img = cv2.resize(img, image_size)
img = np.expand_dims(img, axis=0)
img = preprocess_input(img)
result = model.predict(img, batch_size=1)
print(result)
```

以上代码简单介绍一下：模型测试流程非常清晰，首先加载模型，加载参数>>将数据输入网络>>模型预测。

6 模型保存和导入

```
model = train_model(model, loss, metrics,  optimizer, num_epochs)
os.mkdir('models')
model.save_weights('models/model.h5')
```

模型训练完成后，仅需用 model.save_weights('models/model.h5')一句代码就可以完成模型的保存。同样，模型的导入采用 model.load_weights(model_path, by_name=True)，需要注意的是要设置 by_name=True，这样就能保证和模型名称一样的参数都能加载到模型，当然模型定义要和参数是匹配的，假如要进行 fine-tune 我们只需保证需要重新训练或者新加的网络层的名称和预加载模型参数名称不一样就可以。

7 总结

以上内容涵盖了采用 keras 进行分类任务的全部流程，从数据导入、模型搭建、模型训练、测试，模型保存和导入几个方面分别进行了介绍。当然这只是一些基本的应用，还有一些高级、个性化功能需要我们进一步学习，有机会，下一次介绍一下自定义网络层、设置 check_point、特征可视化等特性。

【mxnet 速成】mxnet 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

这一次我们讲讲 mxnet，相关的代码、数据都在我们 Git 上，希望大家 Follow 一下这个 Git 项目，后面会持续更新不同框架下的任务。

https://github.com/longpeng2008/yousan.ai

1 mxnet 是什么

mxnet 是 amazon 的官方框架，下面参考 mxnet 的官方简介

https://mxnet-bing.readthedocs.io/en/latest/zh/overview.html

深度学习系统通常有两种编程方式，一种是**声明式编程(declarative programming)**，用户只需要声明要做什么，而具体执行则由系统完成。以 Caffe，TensorFlow 的计算图为代表。优点是由于在真正开始计算的时候已经拿到了整个计算图，所以可以做一系列优化来提升性能。实现辅助函数也容易，例如对任何计算图都提供 forward 和 backward 函数，另外也方便对计算图进行可视化，将图保存到硬盘和从硬盘读取。缺点是 debug 很麻烦，监视一个复杂的计算图中的某个节点的中间结果并不简单，逻辑控制也不方便。

一种是**命令式编程(imperative programming)**，以 numpy，torch/pytorch 为代表，每个语句按照原来的意思顺序执行。它 的特点是语义上容易理解，灵活，可以精确控制行为。通常可以无缝地和主语言交互，方便地利用主语言的各类算法，工具包，debug 和性能调试器，但是实现统一的辅助函数和提供整体优化都很困难。

MXNet 尝试将两种模式无缝的结合起来。在命令式编程上 MXNet 提供张量运算，进行模型的迭代训练和更新中的控制逻辑；在声明式编程中 MXNet 支持符号表达式，用来描述神经网络，并利用系统提供的自动求导来训练模型。

随着 pytorch 的崛起，mxnet 已经掉出前三梯队，但不影响喜欢它的人使用。相比于重量级的 tensorflow，mxnet 非常轻量，占用内存少，分布式训练方便，常被用于比赛刷榜（见笔者以前用来刷榜的文）。

如何步入深度学习刷榜第一重境界

2 mxnet 安装配置

喜欢自定义安装和精确控制版本的朋友，可以自行编译，喜欢偷懒的 pip 安装即可，方便快捷。

```
sudo pip install mxnet
```

不过如果你要多机多卡使用，还是源码编译安装吧。

https://github.com/apache/incubator-mxnet

3 mxnet 自定义数据

下面就开始我们的任务，跟以往项目一样，从自定义数据和自定义网络开始。

mxnet 分类任务要求的输入分类文件的格式与 caffe 不一样，为下面的格式，其中分别是序号，标签，路径

```
01../../../../../datas/mouth/1/182smile.jpg
11../../../../../datas/mouth/1/435smile.jpg
```

数据的载入需要用到接口 DataBatch 和 DataIter

https://mxnet.incubator.apache.org/api/python/io/io.html

首先我们定义一下相关的参数配置，主要用于载入训练/测试数据集路径 data-train，data-val，rgb 均值 rgb-mean，类别数目 num-classes 与训练样本集大小 num-examples

```
def add_data_args(parser):
    data = parser.add_argument_group('Data', 'the input images')
    data.add_argument('--data-train', type=str, help='the training data')
    data.add_argument('--data-val', type=str, help='the validation data')
    data.add_argument('--rgb-mean', type=str,
default='123.68,116.779,103.939',help='a tuple of size 3 for the mean
rgb')
    data.add_argument('--pad-size', type=int, default=0,
help='padding the input image')
    data.add_argument('--image-shape', type=str,
help='the image shape feed into the network, e.g. (3,224,224)')
    data.add_argument('--num-classes', type=int,help='the number of
classes')
    data.add_argument('--num-examples', type=int, help='the number of
training examples')
    data.add_argument('--data-nthreads', type=int,
default=4,help='number of threads for data decoding')
    data.add_argument('--benchmark', type=int, default=0,
help='if 1, then feed the network with synthetic data')
    data.add_argument('--dtype', type=str, default='float32',help='data
type: float32 or float16')
```

```
    return data
```

然后，使用 mx.img.ImageIter 来载入图像数据

```
train = mx.img.ImageIter(
        label_width        = 1,
        path_root    = 'data/',
        path_imglist        = args.data_train,
        data_shape         = (3, N_pix, N_pix),
        batch_size         = args.batch_size,
        rand_crop          = True,
        rand_resize        = True,
        rand_mirror        = True,
        shuffle            = True,
        brightness         = 0.4,
        contrast           = 0.4,
        saturation         = 0.4,
        pca_noise          = 0.1,
        num_parts          = nworker,
        part_index         = rank)
```

注意到上面配置了 rand_crop，rand_resize，rand_mirror，shuffle，brightness，contrast，saturation，pca_noise 等选项，这些就是常见的数据增强操作了，如果不懂，可以去看看微信公众号以前的文章

【综述类】一文道尽深度学习中的数据增强方法（上）

【技术综述】深度学习中的数据增强（下）

【开源框架】一文道尽主流开源框架中的数据增强

mxnet 的数据增强接口使用非常的方便，定义如下

```
def add_data_aug_args(parser):
    aug = parser.add_argument_group(
        'Image augmentations', 'implemented in
src/io/image_aug_default.cc')
    aug.add_argument('--random-crop', type=int, default=1,help='if or not
randomly crop the image')
    aug.add_argument('--random-mirror', type=int, default=1,help='if or
not randomly flip horizontally')
    aug.add_argument('--max-random-h', type=int, default=0,help='max
change of hue, whose range is [0, 180]')
```

```
    aug.add_argument('--max-random-s', type=int, default=0,help='max
change of saturation, whose range is [0, 255]')
    aug.add_argument('--max-random-l', type=int, default=0,help='max
change of intensity, whose range is [0, 255]')
    aug.add_argument('--max-random-aspect-ratio', type=float,
default=0,help='max change of aspect ratio, whose range is [0, 1]')
    aug.add_argument('--max-random-rotate-angle', type=int,
default=0,help='max angle to rotate, whose range is [0, 360]')
    aug.add_argument('--max-random-shear-ratio', type=float,
default=0,help='max ratio to shear, whose range is [0, 1]')
    aug.add_argument('--max-random-scale', type=float,
default=1,help='max ratio to scale')
    aug.add_argument('--min-random-scale', type=float,
default=1,help='min ratio to scale, should >= img_size/input_shape.
otherwise use --pad-size')
    return aug
```

可以看到 level >= 1，就可以使用随机裁剪，镜像操作，level >= 2，就可以使用对比度变换操作，level >= 3，就可以使用旋转，缩放等操作。

```
def set_data_aug_level(aug, level):
    if level >= 1:
        aug.set_defaults(random_crop=1, random_mirror=1)
    if level >= 2:
        aug.set_defaults(max_random_h=36, max_random_s=50,
max_random_l=50)
    if level >= 3:
        aug.set_defaults(max_random_rotate_angle=10,
max_random_shear_ratio=0.1, max_random_aspect_ratio=0.25)
```

4 mxnet 网络搭建

同样是三层卷积，两层全连接的网络，话不多说，直接上代码，使用到的 api 是 mxnet.symbol

```
import mxnet as mx

def get_symbol(num_classes, **kwargs):
  if 'use_global_stats' not in kwargs:
```

```
    use_global_stats = False
  else:
    use_global_stats = kwargs['use_global_stats']

  data = mx.symbol.Variable(name='data')
  conv1 = mx.symbol.Convolution(name='conv1', data=data ,
num_filter=12, kernel=(3,3), stride=(2,2), no_bias=True)
  conv1_bn = mx.symbol.BatchNorm(name='conv1_bn', data=conv1 ,
use_global_stats=use_global_stats, fix_gamma=False, eps=0.000100)
  conv1_scale = conv1_bn
  relu1 = mx.symbol.Activation(name='relu1', data=conv1_scale ,
act_type='relu')
  conv2 = mx.symbol.Convolution(name='conv2', data=relu1 ,
num_filter=24, kernel=(3,3), stride=(2,2), no_bias=True)
  conv2_bn = mx.symbol.BatchNorm(name='conv2_bn', data=conv2 ,
use_global_stats=use_global_stats, fix_gamma=False, eps=0.000100)
  conv2_scale = conv2_bn
  relu2 = mx.symbol.Activation(name='relu2', data=conv2_scale ,
act_type='relu')
  conv3 = mx.symbol.Convolution(name='conv3', data=relu2 ,
num_filter=48, kernel=(3,3), stride=(2,2), no_bias=True)
  conv3_bn = mx.symbol.BatchNorm(name='conv3_bn', data=conv3 ,
use_global_stats=use_global_stats, fix_gamma=False, eps=0.000100)
  conv3_scale = conv3_bn
  relu3 = mx.symbol.Activation(name='relu3', data=conv3_scale ,
act_type='relu')
  pool = mx.symbol.Pooling(name='pool', data=relu3 ,
pooling_convention='full', global_pool=True, kernel=(1,1),
pool_type='avg')
  fc = mx.symbol.Convolution(name='fc', data=pool ,
num_filter=num_classes, pad=(0, 0), kernel=(1,1), stride=(1,1),
no_bias=False)
  flatten = mx.symbol.Flatten(data=fc, name='flatten')
  softmax = mx.symbol.SoftmaxOutput(data=flatten, name='softmax')
  return softmax

if __name__ == "__main__":
```

```
net = get_symbol(2) ##二分类任务
net.save('simpleconv3-symbol.json')
```

最后我们可以将其存到 json 文件里，net.save('simpleconv3-symbol.json')，下面是 conv1 的部分，详细大家可以至 git 查看

```
{
  "op": "Convolution",
  "name": "conv1",
  "attr": {
    "kernel": "(3, 3)",
    "no_bias": "True",
    "num_filter": "12",
    "stride": "(2, 2)"
  },
  "inputs": [[0, 0, 0], [1, 0, 0]]
},
```

5 模型训练、测试

5.1 模型训练

准备工作都做好了，训练代码非常简洁，下面就是全部的代码

```
import os
import argparse
import logging
logging.basicConfig(level=logging.DEBUG)
from common import find_mxnet
from common import data, fit
import mxnet as mx

import os, urllib

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="simple conv3 net",

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    train = fit.add_fit_args(parser)
```

```
data.add_data_args(parser)
aug = data.add_data_aug_args(parser)
data.set_data_aug_level(parser, 1)
parser.set_defaults(image_shape='3,48,48', num_epochs=200,
                    lr=.001, wd=0)
args = parser.parse_args()

# define simpleconv3
net = mx.sym.load('models/simple-conv3-symbol.json')
print "net",net

# train
fit.fit(args        = args,
        network     = net,
        data_loader = data.get_rec_iter)
```

其中调用了 fit 接口定义优化目标和策略，我们只分析其中的核心代码，首先是模型创建

```
model = mx.mod.Module(
  context       = devs,
  symbol        = network
)
```

然后是 optimizer 配置，默认使用 adam

```
optimizer_params = {
      'learning_rate': lr,
      'wd' : args.wd
      }
```

初始化

```
initializer = mx.init.Xavier(rnd_type='gaussian', factor_type="in",
magnitude=2.34)
```

最后是完整的接口

```
model.fit(train,
  begin_epoch        = args.load_epoch if args.load_epoch else 0,
  num_epoch          = args.num_epochs,
  eval_data          = val,
  eval_metric        = eval_metrics,
```

```
    kvstore           = kv,
    optimizer         = args.optimizer,
    optimizer_params   = optimizer_params,
    initializer       = initializer,
    arg_params        = arg_params,
    aux_params        = aux_params,
    batch_end_callback = batch_end_callbacks,
    epoch_end_callback = checkpoint,
    allow_missing      = True,
    monitor           = monitor)
```

然后开始愉快的训练吧

```
python train.py --gpus 0 \
  --data-train data/train.txt \
  --model-prefix 'models/simple-conv3' \
  --batch-size 80 --num-classes 2 --num-examples 900 2>&1 | tee log.txt
```

训练模型会存为 simple-conv3-epoch.params 的格式。

5.2 训练过程可视化

由于前面我们的 tensorflow，pytorch，keras 都使用了 tensorborad 进行可视化，mxnet 也可以借助 tensorboard 进行可视化，只需要再设计一些 mxnet 接口即可。具体方法不再赘述，参考 https://github.com/awslabs/mxboard

网络结构的可视化则可用 mx.viz.plot_network(sym).view()。

5.3 模型测试

使用 mx.model.load_checkpoint 载入预训练的模型，如下

```
epoch = int(sys.argv[1]) #check point step
gpu_id = int(sys.argv[2]) #GPU ID for infer
prefix = sys.argv[3]
ctx = mx.gpu(gpu_id)
sym, arg_params, aux_params = mx.model.load_checkpoint(prefix, epoch)
arg_params, aux_params = ch_dev(arg_params, aux_params, ctx)
```

然后使用 bind 接口进行 forward，具体操作如下

```
sym  = mx.symbol.SoftmaxOutput(data = sym, name = 'softmax')
img_full_name = os.path.join(imgdir,imgname)
img = cv2.cvtColor(cv2.imread(img_full_name), cv2.COLOR_BGR2RGB)
```

```
img = np.float32(img)
rows, cols = img.shape[:2]
resize_width = 48
resize_height = 48
img = cv2.resize(img, (resize_width, resize_height),
interpolation=cv2.INTER_CUBIC)
h, w, _ = img.shape

img_crop = img[0:h,0:w] ##此处使用整图
img_crop = np.swapaxes(img_crop, 0, 2)
img_crop = np.swapaxes(img_crop, 1, 2)  # mxnet 的训练是 rgb 的顺序输
入，所以需要调整为 r,g,b 训练
img_crop = img_crop[np.newaxis, :]

arg_params["data"] = mx.nd.array(img_crop, ctx)
arg_params["softmax_label"] = mx.nd.empty((1,), ctx)
exe = sym.bind(ctx, arg_params ,args_grad=None, grad_req="null",
aux_states=aux_params)
exe.forward(is_train=False)
probs = exe.outputs[0].asnumpy()
```

6 总结

好了，就这么多。到今天为止，主流的机器学习框架 caffe，tensorflow，pytorch，paddlepaddle，keras，mxnet 我们已经全部给大家提供了快速入门。

【cntk 速成】cntk 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

欢迎来到专栏《2 小时玩转开源框架系列》，这是我们第七篇，前面已经说过了 caffe，tensorflow，pytorch，mxnet，keras，paddlepaddle。

今天说 cntk，本文所用到的数据，代码请参考我们官方 git

https://github.com/longpeng2008/yousan.ai

1 CNTK 是什么

地址：https://github.com/Microsoft/CNTK

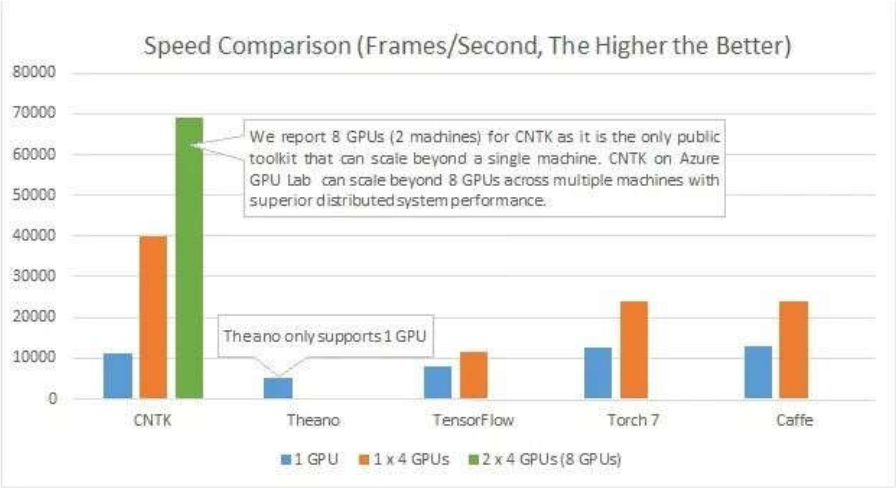
CNTK 是微软开源的深度学习工具包，它通过有向图将神经网络描述为一系列计算步骤。在有向图中，叶节点表示输入值或网络参数，而其他节点表示其输入上的矩阵运算。

CNTK 允许用户非常轻松地实现和组合流行的模型，包括前馈 DNN，卷积网络（CNN）和循环网络（RNN / LSTM）。与目前大部分框架一样，实现了自动求导，利用随机梯度下降方法进行优化。

cntk 有什么特点呢？

1.1 性能较高

按照其官方的说法，比其他的开源框架性能都更高。



笔者在实际进行实验的时候，确实也发现它的训练比较快。

1.2 适合做语音

CNTK 本就是微软语音团队开源的，自然是更合适做语音任务，使用 RNN 等模型，以及在时空尺度分别进行卷积非常容易。

当然，现在的背靠 python 的这些框架已经大同小异，未来实现大一统并非不可能。

2 CNTK 模型训练

pip 安装一条命令即可，可以选择安装 cpu 或者 gpu 版本。

```
pip install cntk/cntk-gpu。
```

接下来就是数据的准备，模型的定义，结果的保存与分析。

在此之前，我们先看官方的分类案例，直观感受一下，代码比较长。

```
from __future__ import print_function
import numpy as np
import cntk as C
from cntk.learners import sgd
from cntk.logging import ProgressPrinter
from cntk.layers import Dense, Sequential
def generate_random_data(sample_size, feature_dim, num_classes):
    # Create synthetic data using NumPy.
    Y = np.random.randint(size=(sample_size, 1), low=0,
high=num_classes)

    # Make sure that the data is separable
    X = (np.random.randn(sample_size, feature_dim) + 3) * (Y + 1)
    X = X.astype(np.float32)
    # converting class 0 into the vector "1 0 0",
    # class 1 into vector "0 1 0", ...
    class_ind = [Y == class_number for class_number in
range(num_classes)]
    Y = np.asarray(np.hstack(class_ind), dtype=np.float32)
    return X, Ydef ffnet():
  inputs = 2
  outputs = 2
  layers = 2
  hidden_dimension = 50
```

```
  # input variables denoting the features and label data
  features = C.input_variable((inputs), np.float32)
  label = C.input_variable((outputs), np.float32)

  # Instantiate the feedforward classification model
  my_model = Sequential ([
              Dense(hidden_dimension, activation=C.sigmoid),
              Dense(outputs)])
  z = my_model(features)

  ce = C.cross_entropy_with_softmax(z, label)
  pe = C.classification_error(z, label)

  # Instantiate the trainer object to drive the model training
  lr_per_minibatch = C.learning_parameter_schedule(0.125)
  progress_printer = ProgressPrinter(0)
  trainer = C.Trainer(z, (ce, pe), [sgd(z.parameters, lr=lr_per_minibatch)],
[progress_printer])

  # Get minibatches of training data and perform model training
  minibatch_size = 25
  num_minibatches_to_train = 1024

  aggregate_loss = 0.0
  for i in range(num_minibatches_to_train):
      train_features, labels = generate_random_data(minibatch_size, inputs,
outputs)
      # Specify the mapping of input variables in the model to actual
minibatch data to be trained with
      trainer.train_minibatch({features : train_features, label : labels})
      sample_count = trainer.previous_minibatch_sample_count
      aggregate_loss += trainer.previous_minibatch_loss_average *
sample_count

  last_avg_error = aggregate_loss / trainer.total_number_of_samples_seen

  test_features, test_labels = generate_random_data(minibatch_size,
```

```
inputs, outputs)
  avg_error = trainer.test_minibatch({features : test_features, label :
test_labels})
  print(' error rate on an unseen minibatch: {}'.format(avg_error))
  return last_avg_error, avg_errornp.random.seed(98052)ffnet()
```

上面就是一个两层的全连接神经网络，使用 input_variable 封装数据，使用 Sequential 定义模型，使用 train_minibatch({features : train_features, label : labels})来 feed 数据，与 tf，pytorch 等框架都是一样的，的确是没有什么好说的。

2.1 数据读取

这里需要用到接口，io.ImageDeserializer 与 C.io.StreamDefs，C.io.StreamDef。

它可以直接输入如下格式的 txt 文件用于图像分类问题。

```
../../../../datas/mouth/1/182smile.jpg1
../../../../datas/mouth/1/435smile.jpg1
../../../../datas/mouth/0/40neutral.jpg0
../../../../datas/mouth/1/206smile.jpg1
```

注意上面采用的分隔符是'\t'，这一点与 MXNet 相同，与 caffe 不同，完整的解析代码如下：

```
C.io.MinibatchSource(C.io.ImageDeserializer(map_file, C.io.StreamDefs(
features = C.io.StreamDef(field='image', transforms=transforms),
labels   = C.io.StreamDef(field='label', shape=num_classes)
)))
```

在对图像数据进行封装的时候，添加了 transform，所以可以在这里进行数据预处理操作。

常用的裁剪与缩放如下：

```
transform.crop(crop_type='randomside', side_ratio=0.8)
transform.scale(width=image_width, height=image_height,
channels=num_channels, interpolations='linear')
```

C.io.MinibatchSource 的返回就是数据指针，可以直接用于训练。

2.2 网络定义

与 tensorflow 和 pytorch 颇为相似，如下

```
def simpleconv3(input, out_dims):
  with C.layers.default_options(init=C.glorot_uniform(), activation=C.relu):
```

```
    net = C.layers.Convolution((3,3), 12, pad=True)(input)
    net = C.layers.MaxPooling((3,3), strides=(2,2))(net)

    net = C.layers.Convolution((3,3), 24, pad=True)(net)
    net = C.layers.MaxPooling((3,3), strides=(2,2))(net)

    net = C.layers.Convolution((3,3), 48, pad=True)(net)
    net = C.layers.MaxPooling((3,3), strides=(2,2))(net)

    net = C.layers.Dense(128)(net)
    net = C.layers.Dense(out_dims, activation=None)(net)

  return net
```

2.3 损失函数与分类错误率指标定义

如下，model_func 就是上面的 net，input_var_norm 和 label_var 分别就是数据和标签。

```
z = model_func(input_var_norm, out_dims=2)
ce = C.cross_entropy_with_softmax(z, label_var)
pe = C.classification_error(z, label_var)
```

2.4 训练参数

就是学习率，优化方法，epoch 等配置。

```
epoch_size     = 900
minibatch_size = 64
lr_per_minibatch       = C.learning_rate_schedule([0.01]*100 + [0.003]*100
+ [0.001],
C.UnitType.minibatch, epoch_size)
m = C.momentum_schedule(0.9)
l2_reg_weight          = 0.001
learner = C.momentum_sgd(z.parameters,
lr = lr_per_minibatch,
momentum = m,
l2_regularization_weight=l2_reg_weight)
progress_printer = C.logging.ProgressPrinter(tag='Training',
num_epochs=max_epochs)
trainer = C.Trainer(z, (ce, pe), [learner], [progress_printer])
```

注意学习率的配置比较灵活，通过 learning_rate_schedule 接口，上面的 C.learning_rate_schedule([0.01]*100 + [0.003]*100 + [0.001]意思是，在 0～100 epoch，使用 0.01 的学习率，100～100+100 epoch，使用 0.003 学习率，此后使用 0.001 学习率。

2.5 训练与保存

使用数据指针的 next_minibatch 获取训练数据，trainer 的 train_minibatch 进行训练，可以看出 cntk 非常强调 minibatch 的概念，实际上学习率和优化方法都可以针对单个样本进行设置。

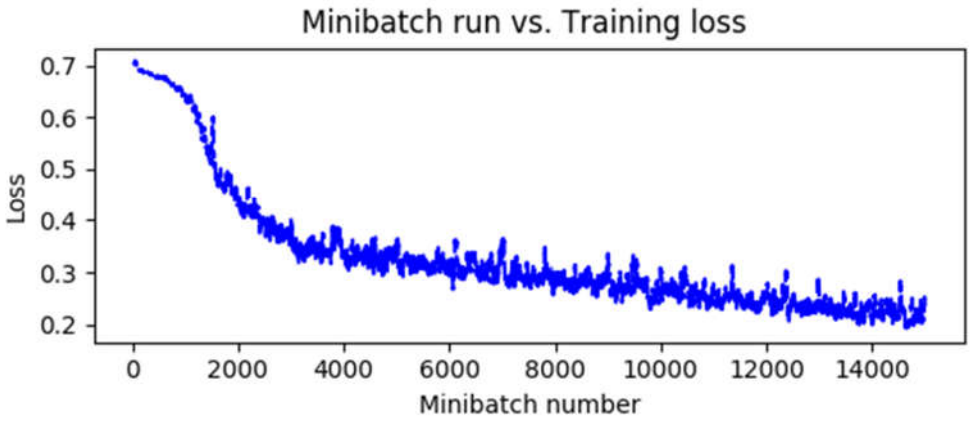
```
for epoch in range(max_epochs):
  sample_count = 0
  while sample_count < epoch_size:
    data = reader_train.next_minibatch(min(minibatch_size, epoch_size - sample_count), input_map=input_map)
    trainer.train_minibatch(data)
```

模型的保存就一行代码：

```
z.save("simpleconv3.dnn")
```

2.6 可视化

需要可视化的内容不多，就是 loss 曲线和精度曲线，所以可以直接自己添加代码，用上面的模型训练最后的 loss 如下，更好参数可自己调。



3 CNTK 模型测试

测试就是载入模型，做好与训练时同样的预处理操作然后 forward 就行了。

```
import ***
model_file = sys.argv[1]
image_list = sys.argv[2]
model = C.load_model(model_file)

count = 0
acc = 0
imagepaths = open(image_list,'r').readlines()
for imagepath in imagepaths:
  imagepath,label = imagepath.strip().split('\t')
  im = Image.open(imagepath)
  print imagepath
  print "im size",im.size
  image_data = np.array(im,dtype=np.float32)
  image_data = cv2.resize(image_data,(image_width,image_height))
  image_data = np.ascontiguousarray(np.transpose(image_data, (2, 0, 1)))
  output = model.eval({model.arguments[0]:[image_data]})[0]
  print output
  print label,np.argmax(np.squeeze(output))
  if str(label) == str(np.argmax(np.squeeze(output))):
     acc = acc + 1
  count = count + 1
print "acc=",float(acc) / float(count)
```

最终模型训练集准确率 91%，测试集准确率 88%，大家可以自己去做更多调试。

4 总结

相比于 tensorflow，pytorch，cntk 固然是比较小众，但也不失为一个优秀的平台，尤其是对于语音任务，感兴趣大家可以自行体验，代码已经上传至 https://github.com/longpeng2008/yousan.ai。

【chainer 速成】chainer 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

欢迎来到专栏《2 小时玩转开源框架系列》，这是我们第八篇，前面已经说过了 caffe，tensorflow，pytorch，mxnet，keras，paddlepaddle，cntk。

今天说 chainer，本文所用到的数据，代码请参考我们官方 git

https://github.com/longpeng2008/yousan.ai

1 chainer 是什么

chainer 是一个基于 python 的深度学习框架，能够轻松直观地编写复杂的神经网络架构。

当前大多数深度学习框架都基于**“Define-and-Run”**方案。也就是说，首先定义网络，然后用户定期向其提供小批量的训练数据。由于网络静态定义的，因此所有的逻辑必须作为数据嵌入到网络架构中。

相反，chainer 采用“Define-by-Run”方案，即通过实际的前向计算动态定义网络。更确切地说，chainer 存储计算历史而不是编程逻辑。这样，Chainer 不需要将条件和循环引入网络定义 。chainer 的核心理念就是 Define-by-Run。

2 chainer 训练准备

2.1 chainer 安装

chainer 安装很简单，只需要在终端输入下面命令即可安装：

```
pip install chainer
```

2.2 数据读取

在 chainer 中读取数据是非常简单的。数据读取部分的代码如下：

```
import numpy as np
import os
from PIL import Image
import glob
from chainer.datasets import tuple_dataset
class Dataset():
    def __init__(self, path, width=60, height=60):
        channels = 3
        path = glob.glob('./mouth/*')
        pathsAndLabels = []
        index = 0
        for p in path:
            print(p + "," + str(index))
            pathsAndLabels.append(np.asarray([p, index]))
            index = index + 1
        allData = []
        for pathAndLabel in pathsAndLabels:
            path = pathAndLabel[0]
            label = pathAndLabel[1]
            imagelist = glob.glob(path + "/*")
            for imgName in imagelist:
                allData.append([imgName, label])
        allData = np.random.permutation(allData)
        imageData = []
        labelData = []
```

下面解释下在 chainer 中读取数据的一些特色，完整代码请移步 github。

在 chainer 中我们通过 chainer.datasets 模块来获取数据集，其最基本的数据集就是一个数组，平时最常见的 NumPy 和 CuPy 数组都可以直接用作数据集。在本实例中我们采用的是元组数据集即 TupleDataset()来获取数据。

2.3 网络定义

它的网络定义和 pytorch 基本上是相似的，如下：

```
class MyModel(Chain):
    def __init__(self):
        super(MyModel, self).__init__()
        with self.init_scope():
```

```
        self.conv1 = L.Convolution2D(
            in_channels=3, out_channels=12, ksize=3, stride=2)
        self.bn1 = L.BatchNormalization(12)
        self.conv2 = L.Convolution2D(
            in_channels=12, out_channels=24, ksize=3, stride=2)
        self.bn2 = L.BatchNormalization(24)
        self.conv3 = L.Convolution2D(
            in_channels=24, out_channels=48, ksize=3, stride=2)
        self.bn3 = L.BatchNormalization(48)
        self.fc1 = L.Linear(None, 1200)
        self.fc2 = L.Linear(1200, 128)
        self.fc3 = L.Linear(128, 2)
    def __call__(self,x):
        return self.forward(x)
    def forward(self, x):
        h1 = F.relu(self.conv1(x))
        h2 = F.relu(self.conv2(h1))
        h3 = F.relu(self.conv3(h2))
        h4 = F.relu(self.fc1(h3))
        h5 = F.relu(self.fc2(h4))
        x = self.fc3(h5)
        return (x)
```

上面的例子和之前说过的caffe、tensorflow、pytorch等框架采用的网络结构是一样。这里不在赘述，我具体说下这个框架的特色。

(1) MyModel(Chain)

Chain 在 chainer 中是一个定义模型的类，我们把模型 MyModel 定义为 Chain 的子类，即继承 Chain 这个类，这和 Pytorch 中的 nn.module 类似。以后我们在模型定义时都可以通过 Chain 来构建具有潜在深层功能和链接层次的模型。

(2) Link 和 Function

在 Chainer 中，神经网络的每一层都可以认为是由两种广泛类型的函数之一组成即 Link 和 Function。

其中 Function 是一个没有可学习参数的函数，而 LInk 是包括参数的，我们也能把 Link 理解成一个赋予其参数的 Function。

在我们使用它之前，我们首先需要导入相应的模块，如下：

```
import chainer.links as L
import chainer.functions as F
```

另外在平时使用时我们喜欢用 L 替代 Link，用 F 代替 Function。如 L.Convolution2D 和 F.relu

(3) __call__

对于__call__它的作用就是使我们的 chain 像一个函数一样容易被调用。

3 模型训练

数据加载和网络定义好后，我们就可以进行模型训练了，话不多说，我们直接上代码。

```
model = L.Classifier(MyModel())
if os.path.isfile('./dataset.pickle'):
    print("dataset.pickle is exist. loading...")
    with open('./dataset.pickle', mode='rb') as f:
        train, test = pickle.load(f)
        print("Loaded")
    else:
        datasets = dataset.Dataset("mouth")
        train, test = datasets.get_dataset()
        with open('./dataset.pickle', mode='wb') as f:
            pickle.dump((train, test), f)
            print("saving train and test...")
    optimizer = optimizers.MomentumSGD(lr=0.001, momentum=0.5)
    optimizer.setup(model)
    train_iter = iterators.SerialIterator(train, 64)
    test_iter = iterators.SerialIterator(test, 64, repeat=False, shuffle=True)
    updater = training.StandardUpdater(train_iter, optimizer, device=-1)
    trainer = training.Trainer(updater, (800, 'epoch'),
out='{}_model_result'.format(MyModel.__class__.__name__))
```

在 chainer 中，模型训练可以分为如下 6 个步骤，个人认为这 6 个步骤是非常好理解的。

Step-01-Dataset

第一步当然就是加载我们的数据集了，我们通常都是通过下面方法加载数据集：

```
train, test = datasets.get_dataset()
```

Step-02-Iterator

chainer 提供了一些 Iterator，通常我们采用下面的方法来从数据集中获取小批量的数据进行迭代。

```
train_iter = iterators.SerialIterator(train, batchsize)
test_iter = iterators.SerialIterator(test, batchsize, repeat=False,
shuffle=True)
```

Step-03-Model

在 chainer 中 chainer.links.Classifier 是一个简单的分类器模型，尽管它里面有许多参数如 predictor、lossfun 和 accfun，但我们只需赋予其一个参数那就是 predictor，即你定义过的模型。

```
model = L.Classifier(MyModel())
```

Step-04-Optimizer

模型弄好后，接下来当然是优化了，在 chainer.optimizers 中有许多我们常见的优化器，部分优化器如下：

1、chainer.optimizers.AdaDelta

2、chainer.optimizers.AdaGrad

3、chainer.optimizers.AdaDelta

4、chainer.optimizers.AdaGrad

5、chainer.optimizers.Adam

6、chainer.optimizers.CorrectedMomentumSGD .

7、chainer.optimizers.MomentumSGD

8、chainer.optimizers.NesterovAG

9、chainer.optimizers.RMSprop

10、chainer.optimizers.RMSpropGraves

...

Step-05-Updater

当我们想要训练神经网络时，我们必须运行多次更新参数，这在 chainer 中就是 Updater 所做的工作，在本例我们使用的是 training.StandardUpdater。

Step-06-Trainer

上面的工作做完之后我们需要做的就是训练了。在 chainer 中，训练模型采用的是 training.Trainer()。

4 可视化

```
trainer.extend(extensions.dump_graph("main/loss"))
trainer.extend(extensions.Evaluator(test_iter, model, device=-1))
trainer.extend(extensions.LogReport())
```

```
trainer.extend(extensions.PrintReport( ['epoch', 'main/loss',
'validation/main/loss', 'main/accuracy', 'validation/main/accuracy']))
trainer.extend(extensions.PlotReport(['main/loss', 'validation/main/loss'],
x_key='epoch', file_name='loss.png'))
trainer.extend(extensions.PlotReport(['main/accuracy',
'validation/main/accuracy'], x_key='epoch', file_name='accuracy.png'))
trainer.extend(extensions.ProgressBar())
```

在 chainer 中可视化是非常方便的，我们常通过 trainer.extend()来实现我们的可视化，其有下面几种可视化的方式。

1、chainer.training.extensions.PrintReport

2、chainer.training.extensions.ProgressBar

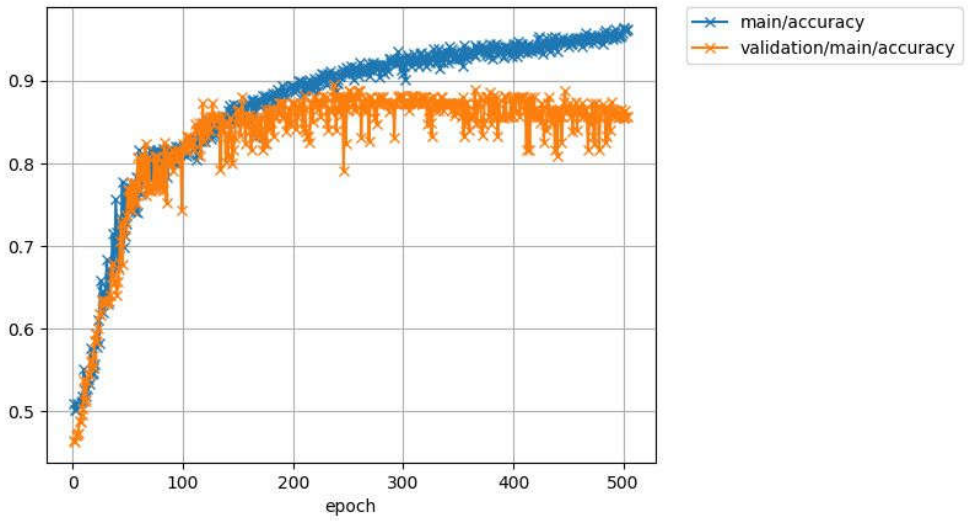
3、chainer.training.extensions.LogReport

4、chainer.training.extensions.PlotReport

5、chainer.training.extensions.VariableStatisticsPlot

6、chainer.training.extensions.dump_graph

以上就是利用 chain 来做一个图像分类任务的一个小例子。完整代码可以看配套的 git 项目，我们看看训练中的记录，如下：



5 总结

本文讲解了如何使用 chainer 深度学习框架完成一个分类任务，尽管这个框架用的人不多，但这个框架使用起来还是比较方便的，您在用吗？如果您在用，可以联系我们一起交流下！

【DL4J 速成】Deeplearning4j 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

欢迎来到专栏《2 小时玩转开源框架系列》，这是我们第九篇，前面已经说过了 caffe，tensorflow，pytorch，mxnet，keras，paddlepaddle，cntk，chainer。

今天说 Deeplearning4j(DL4J)，本文所用到的数据，代码请参考我们官方 git

https://github.com/longpeng2008/yousan.ai

1 Deeplearning4j(DL4J)是什么

不同于深度学习广泛应用的语言 Python，DL4J 是为 **java 和 jvm** 编写的开源深度学习库，支持各种深度学习模型。

DL4J 最重要的特点是支持分布式，可以在 Spark 和 Hadoop 上运行，支持分布式 CPU 和 GPU 运行。DL4J 是为商业环境，而非研究所设计的，因此更加贴近某些生产环境。

2 DL4J 训练准备

2.1 DL4J 安装

系统要求:

Java：开发者版 7 或更新版本（仅支持 64 位版本）

Apache Maven：Maven 是针对 Java 的项目管理工具，兼容 IntelliJ 等 IDE，可以让我们轻松安装 DL4J 项目库

IntelliJ IDEA (建议)或 Eclipse Git

官方提供了很多 DL4J 的示例。可以通过以下命令下载安装：

```
$ git clone https://github.com/deeplearning4j/dl4j-examples.git
$ cd dl4j-examples/
$ mvn clean install
```

mvn clean install 目的是为了安装所依赖的相关包。

然后将下载的 dl4j-examples 导入到 IntelliJ IDEA 中，点击自己想要试的例子进行运行。

2.2 数据准备

DL4J 有自己的特殊的数据结构 DataVec，所有的输入数据在进入神经网络之前要先经过向量化。向量化后的结果就是**一个行数不限的单列矩阵**。

熟悉 Hadoop/MapReduce 的朋友肯定知道它的输入用 InputFormat 来确定具体的 InputSplit 和 RecordReader。DataVec 也有自己 FileSplit 和 RecordReader，并且对于不同的数据类型（文本、CSV、音频、图像、视频等），有不同的 RecordReader，下面是一个图像的例子。

```
int height = 48;  // 输入图像高度
int width = 48;   // 输入图像宽度
int channels = 3; // 输入图像通道数
int outputNum = 2; // 2 分类
int batchSize = 64;
int nEpochs = 100;
int seed = 1234;
Random randNumGen = new Random(seed);

// 训练数据的向量化
File trainData = new File(inputDataDir + "/train");
FileSplit trainSplit = new FileSplit(trainData,
NativeImageLoader.ALLOWED_FORMATS, randNumGen);
ParentPathLabelGenerator labelMaker = new ParentPathLabelGenerator();
// parent path as the image label
ImageRecordReader trainRR = new ImageRecordReader(height, width,
channels, labelMaker);
trainRR.initialize(trainSplit);
DataSetIterator trainIter = new RecordReaderDataSetIterator(trainRR,
batchSize, 1, outputNum);

// 将像素从 0-255 缩放到 0-1 (用 min-max 的方式进行缩放)
DataNormalization scaler = new ImagePreProcessingScaler(0, 1);
scaler.fit(trainIter);
trainIter.setPreProcessor(scaler);

// 测试数据的向量化
File testData = new File(inputDataDir + "/test");
```

```
FileSplit testSplit = new FileSplit(testData,
NativeImageLoader.ALLOWED_FORMATS, randNumGen);
ImageRecordReader testRR = new ImageRecordReader(height, width,
channels, labelMaker);
testRR.initialize(testSplit);
DataSetIterator testIter = new RecordReaderDataSetIterator(testRR,
batchSize, 1, outputNum);
testIter.setPreProcessor(scaler); // same normalization for better results
```

数据准备的过程分成以下几个步骤：

1）通过 FileSplit 处理输入文件，FileSplit 决定了文件的分布式的分发和处理。

2）ParentPathLabelGenerator 通过父目录来直接生成标签，这个生成标签的接口非常方便，比如说如果是二分类，我们先将两个父目录设定为 0 和 1，然后再分别在里面放置对应的图像就行。

3）通过 ImageRecordReader 读入输入图像。RecordReader 是 DataVec 中的一个类，ImageRecordReader 是 RecordReader 中的一个子类，这样就可以将输入图像转成向量化的带有索引的数据。

4）生成 DataSetIterator，实现了对输入数据集的迭代。

2.3 网络定义

在 Deeplearning4j 中，添加一个层的方式是通过 NeuralNetConfiguration.Builder() 调用 layer，指定其在所有层中的输入及输出节点数 nIn 和 nOut，激活方式 activation，层的类型如 ConvolutionLayer 等。

```
// 设置网络层及超参数
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
  .seed(seed)
  .l2(0.0005)
  .updater(new Adam(0.0001))
  .weightInit(WeightInit.XAVIER)
  .list()
  .layer(0, new ConvolutionLayer.Builder(3, 3)
      .nIn(channels)
      .stride(2, 2)
      .nOut(12)
      .activation(Activation.RELU)
      .weightInit(WeightInit.XAVIER)
```

```
        .build())
    .layer(1, new BatchNormalization.Builder()
        .nIn(12)
        .nOut(12)
        .build())
    .layer(2, new ConvolutionLayer.Builder(3, 3)
        .nIn(12)
        .stride(2, 2)
        .nOut(24)
        .activation(Activation.RELU)
        .weightInit(WeightInit.XAVIER)
        .build())
    .layer(3, new BatchNormalization.Builder()
        .nIn(24)
        .nOut(24)
        .build())
    .layer(4, new ConvolutionLayer.Builder(3, 3)
        .nIn(24)
        .stride(2, 2)
        .nOut(48)
        .activation(Activation.RELU)
        .weightInit(WeightInit.XAVIER)
        .build())
    .layer(5, new BatchNormalization.Builder()
        .nIn(48)
        .nOut(48)
        .build())
    .layer(6, new DenseLayer.Builder().activation(Activation.RELU)
        .nOut(128).build())
    .layer(7, new
OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOO
D)
        .nOut(outputNum)
        .activation(Activation.SOFTMAX)
        .build())
```

```
  // InputType.convolutional for normal image
  .setInputType(InputType.convolutionalFlat(48, 48, 3))
  .backprop(true).pretrain(false).build();
```

这里的网络结构和之前的 caffe、tensorflow、pytorch 等框架采用的网络结构是一样的，都是一个 3 层的神经网络。

3 模型训练

数据准备好了，网络也建好了，接下来就可以训练了。

```
// 新建一个多层网络模型
MultiLayerNetwork net = new MultiLayerNetwork(conf);
net.init();
// 训练的过程中同时进行评估
for (int i = 0; i < nEpochs; i++) {
  net.fit(trainIter);
  log.info("Completed epoch " + i);
  Evaluation trainEval = net.evaluate(trainIter);
  Evaluation eval = net.evaluate(testIter);
  log.info("train: " + trainEval.precision());
  log.info("val: " + eval.precision());
  trainIter.reset();
  testIter.reset();
}
//保存模型
ModelSerializer.writeModel(net, new File(modelDir + "/mouth-
model.zip"), true);
```

训练的过程非常简单直观，直接通过 **net.fit()**加载 trainIter 就可以，其中 trainIter 在数据准备中已经定义好了。

通过 **net.evaluate(trainIter)和 net.evaluate(testIter)**的方式来评估训练和测试的表现，这里我们将每个 epoch 的准确率打印出来。

4 可视化

DL4J 提供的用户界面可以在浏览器中看到实时的训练过程。

第一步：

将用户界面依赖项添加到 pom 文件中：

```
<dependency>
      <groupId>org.deeplearning4j</groupId>
      <artifactId>deeplearning4j-ui_2.10</artifactId>
      <version>${dl4j.version}</version>
  </dependency>
```

第二步：

在项目中启动用户界面

```
//初始化用户界面后端,获取一个 UI 实例
UIServer uiServer = UIServer.getInstance();
//设置网络信息（随时间变化的梯度、分值等）的存储位置。这里将其存储于内存。
StatsStorage statsStorage = new InMemoryStatsStorage();
//将 StatsStorage 实例连接至用户界面，让 StatsStorage 的内容能够被可视化
uiServer.attach(statsStorage);
//添加 StatsListener 来在网络定型时收集这些信息
net.setListeners(new StatsListener(statsStorage));
```

首先我们初始化一个用户界面后端，设置网络信息的存储位置。

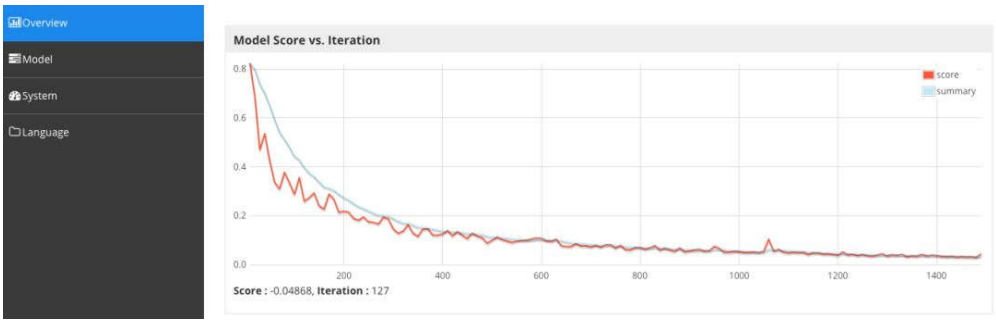
这里将其存储于内存，也可以放入文件中，通过 new FileStatsStorage(File)的方式实现。

再将 StatsStorage 实例连接至用户界面，让 StatsStorage 的内容能够被可视化。

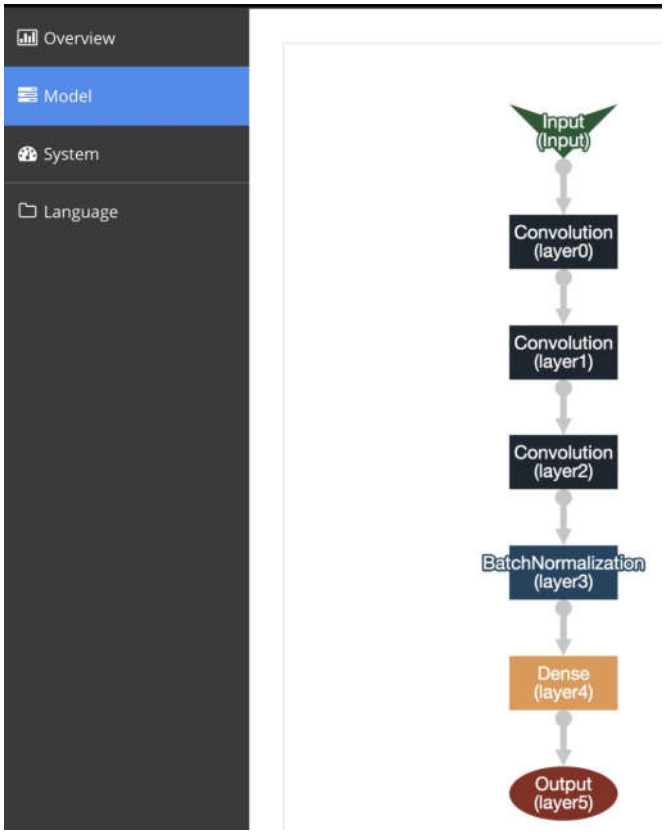
最后添加 StatsListener 监听，在网络定型时收集这些信息。

默认的浏览器地址是：http://localhost:9000/train/overview

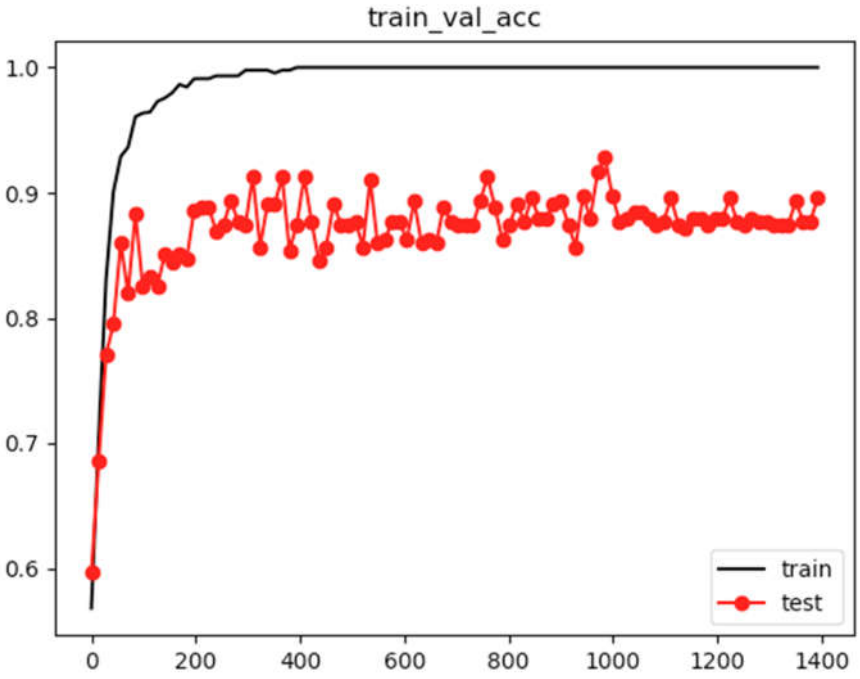
下面可视化一下损失函数值随迭代次数的变化曲线



模型页面中可以直观感受我们建立的模型



看一下最后的训练集和测试集的准确率



有一些过拟合，主要原因还是数据太少。

以上就是我们用自己的数据在 DL4J 框架上实践的内容，完整代码可以参考官方 git。

5 总结

本文讲解了如何使用DL4J深度学习框架完成一个分类任务，虽然这个框架不是很热门，但是它是唯一集成 java 和大数据平台的，您在用吗？如果您在用，可以联系我们一起交流下！另外，还有想让我们介绍的框架吗？欢迎留言。

【MatConvnet 速成】MatConvnet 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

欢迎来到专栏《2 小时玩转开源框架系列》，这是我们第 10 篇，前面已经说过了 caffe，tensorflow，pytorch，mxnet，keras，paddlepaddle，cntk，chainer，deeplearning4j。

本文说下 MatConvnet，本文所用到的数据，代码请参考我们官方 git

https://github.com/longpeng2008/yousan.ai

1 MatConvnet 是什么

不同于各类深度学习框架广泛使用的语言 Python，MatConvnet 是用 matlab 作为接口语言的开源深度学习库，底层语言是 cuda。

官网地址为：http://www.vlfeat.org/matconvnet/

github 地址为：https://github.com/vlfeat/matconvnet

因为是在 matlab 下面，所以 debug 的过程非常的方便，而且本身就有很多的研究者一直都使用 matlab 语言，所以其实该语言的群体非常大。

在用 python 之前，我也是用 matlab 的，那个经典的 deep-learning-toolbox 的代码其实也非常值得研读，说起来，matlab 还是非常做图像处理的。

2 MatConvnet 训练准备

2.1 安装

以 linux 系统为例，首先要安装好 matlab，这个大家自己搞定吧。然后，在 matlab 环境下进行安装，几行代码就可以。

```
mex -setup ##设置好编译器
untar('http://www.vlfeat.org/matconvnet/download/matconvnet-1.0-beta25.tar.gz') ;
cd matconvnet-1.0-beta25
run matlab/vl_compilenn ;
```

没有报错的话就完成了，完成后为了确保没有问题，先用官方的例子确认一下。

```
%下载预训练模型
```

```
urlwrite(...  'http://www.vlfeat.org/matconvnet/models/imagenet-vgg-
f.mat', ...  'imagenet-vgg-f.mat')
%设置环境
run matlab/vl_setupnn

%载入模型
net = load('imagenet-vgg-f.mat') ;
net = vl_simplenn_tidy(net)

%读取图像并预处理
im=imread('peppers.png');
im_ = single(im) ; % note: 255 range
im_=imresize(im_,net.meta.normalization.imageSize(1:2));
im_ = im_ - net.meta.normalization.averageImage ;

%得到结果
res = vl_simplenn(net, im_) ;
result.scores=squeeze(gather(res(end).x));
[bestScore,best]=max(scores);
figure(1) ; clf ; imagesc(im) ;
title(sprintf('%s(%d),score%.3f',...
  net.meta.classes.description{best}, best, bestScore)) ;
```

成功的话结果如下：

更复杂的还可以用DagNN wrapper的API，不过这不是本文的主要目标，因此不再讲述。

当然，我们是要用 GPU 的，所以还要完成 GPU 编译，按照这里来：

http://www.vlfeat.org/matconvnet/install/

因为版本不一定完全匹配，所以用 nvcc 编译。

```
vl_compilenn('enableGpu', true, 'cudaRoot',
'/usr/local/cuda','cudaMethod', 'nvcc')
```

2.2 数据准备

前面讲了官方的例子，接下来就是要用我们自己的例子了，第一步还是老规矩，准备数据，完整的代码如下。

```
function imdb = mydataset(datadir)
inputSize =[48,48,1];
subdir=dir(datadir);
imdb.images.data=[];
imdb.images.labels=[];
imdb.images.set = [] ;
imdb.meta.sets = {'train', 'val', 'test'} ;
image_counter=0;
trainratio=0.8;
subdir
for i=3:length(subdir)
    imgfiles=dir(fullfile(datadir,subdir(i).name));
    imgpercategory_count=length(imgfiles)-2;
    disp([i-2 imgpercategory_count]);
    image_counter=image_counter+imgpercategory_count;
    for j=3:length(imgfiles)
        img=imread(fullfile(datadir,subdir(i).name,imgfiles(j).name));
        img=imresize(img, inputSize(1:2));
        img=single(img);
%         [~,~,d]=size(img);
%         if d==3
%             img=rgb2gray(img);
%             continue;
%         end
        imdb.images.data(:,:,:,end+1)=single(img);
```

```
        imdb.images.labels(end+1)= i-2;
        if j-2<imgpercategory_count*trainratio
           imdb.images.set(end+1)=1;
        else
           imdb.images.set(end+1)=3;
        end
      end
end
dataMean=mean(imdb.images.data,4);
imdb.images.data = single(bsxfun(@minus,imdb.images.data,
dataMean)) ;
imdb.images.data_mean = dataMean;
end
```

如果使用过 Matlab 的同学，应该一下就看懂了，实际上就是 3 个步骤：

1）使用 fullfile 函数遍历图像。

2）预处理，包括缩放，类型转换等。

3）生成 IMDB 格式数据集。

2.3 网络定义

还是跟以前一样，定义一个 3 层的卷积神经网络，非常简单，不做过多注释了噢。

```
Function net =simpleconv3()
rng('default');
rng(0) ;

f=1/100 ;
usebatchNormalization = true ;

net.layers = {};
net.layers{end+1} = struct('type', 'conv', ...
                    'weights', {{f*randn(3,3,3,12, 'single'), zeros(1, 12,
'single')}}, ...
                    'stride', 1, ...
                    'pad', 1) ;
net.layers{end+1} = struct('type', 'pool', ...
                    'method', 'max', ...
                    'pool', [2 2], ...
```

```
                    'stride', 2, ...
                    'pad', 0) ;
net.layers{end+1} = struct('type', 'relu') ;

net.layers{end+1} = struct('type', 'conv', ...
                    'weights', {{f*randn(3,3,12,24,
'single'),zeros(1,24,'single')}}, ...
                    'stride', 1, ...
                    'pad', 1) ;
net.layers{end+1} = struct('type', 'pool', ...
                    'method', 'max', ...
                    'pool', [2 2], ...
                    'stride', 2, ...
                    'pad', 0) ;
net.layers{end+1} = struct('type', 'relu') ;

net.layers{end+1} = struct('type', 'conv', ...
                    'weights', {{f*randn(3,3,24,48,
'single'),zeros(1,48,'single')}}, ...
                    'stride', 1, ...
                    'pad', 1) ;
net.layers{end+1} = struct('type', 'pool', ...
                    'method', 'max', ...
                    'pool', [2 2], ...
                    'stride', 2, ...
                    'pad', 0) ;
net.layers{end+1} = struct('type', 'relu') ;

net.layers{end+1} = struct('type', 'conv', ...
 'weights', {{f*randn(6,6,48,2, 'single'),zeros(1,2,'single')}}, ...
 'stride', 1, ...
 'pad', 0) ;


net.layers{end+1} = struct('type', 'softmaxloss') ;

net = insertBnorm(net, 1) ;
```

```
net = insertBnorm(net, 5) ;
net = insertBnorm(net, 9) ;

% Meta parameters
net.meta.inputSize = [48 48 3] ;
net.meta.trainOpts.learningRate = logspace(-2, -5, 100);
net.meta.trainOpts.numEpochs = 50 ;
net.meta.trainOpts.batchSize = 16 ;

% Fill in defaul values
net = vl_simplenn_tidy(net) ;

end

% --------------------------------------------------------------------
```

3 模型训练

完整代码如下。

```
function [net, info] = trainconv3()
global datadir;
run matlab/vl_setupnn ; %初始化

datadir='/home/longpeng/project/LongPeng_ML_Course/projects/classifi
cation/matconvnet/conv3/mouth';
opts.expDir =
fullfile('/home/longpeng/project/LongPeng_ML_Course/projects/classifica
tion/matconvnet/conv3/','imdb') ;
opts.imdbPath = fullfile(opts.expDir, 'imdb.mat');

if exist(opts.imdbPath,'file')
  imdb=load(opts.imdbPath);
else
  imdb=mydataset(datadir);
  mkdir(opts.expDir) ;
  save(opts.imdbPath, '-struct', 'imdb') ;
```

```
end

net=simpleconv3();
net.meta.normalization.averageImage =imdb.images.data_mean ;
opts.train.gpus=1;

[net, info] = cnn_train(net, imdb, getBatch(opts), ...
 'expDir', opts.expDir, ...
 net.meta.trainOpts, ...
 opts.train, ...
 'val', find(imdb.images.set == 3)) ;

function fn = getBatch(opts)
% --------------------------------------------------------------------
  fn = @(x,y) getSimpleNNBatch(x,y) ;
end

function [images, labels]  = getSimpleNNBatch(imdb, batch)
  images = imdb.images.data(:,:,:,batch) ;
  labels = imdb.images.labels(1,batch) ;
  if opts.train.gpus > 0
     images = gpuArray(images) ;
  end
end
```

总共就这么几个步骤：

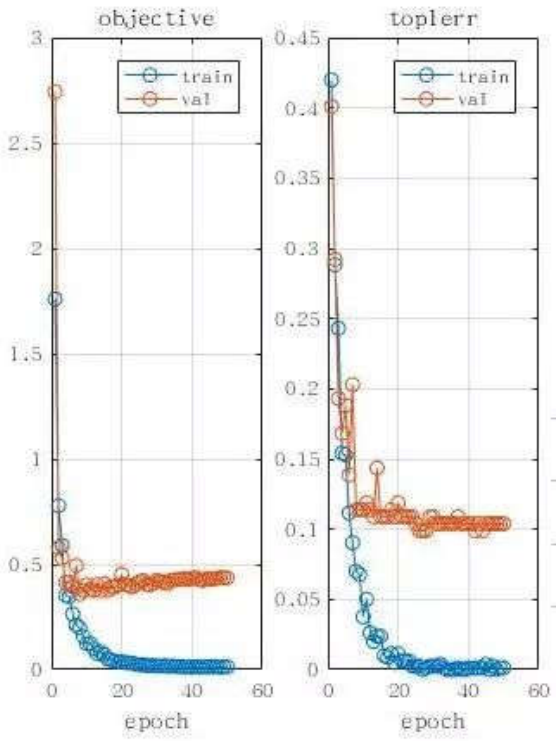
1）初始化环境，run matlab/vl_setupnn 。

2）定义网络：net=simpleconv3()。

3）调用训练接口：[net, info] = cnn_train(net, imdb, getBatch(opts)。

4 可视化

工具已经给封装好了可视化，直接运行代码就会跳出来，可以看出收敛正常，略有过拟合，展示的分别是 softmax 损失和错误率。



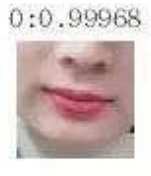
5 测试

```
%netpath=[opts.expDir '/net-epoch-50.mat'];
netpath='/home/longpeng/project/LongPeng_ML_Course/projects/classifi
cation/matconvnet/conv3/imdb/net-epoch-50.mat';
class=1;index=1;
datadir='/home/longpeng/project/LongPeng_ML_Course/projects/classifi
cation/matconvnet/conv3/mouth';
subdir=dir(datadir);
imgfiles=dir(fullfile(datadir,subdir(class+2).name));
img=imread(fullfile(datadir,subdir(class+2).name,imgfiles(index+2).name)
);
imshow(img);
net=load(netpath);
net=net.net;
im_=single(img);
im_=imresize(im_,net.meta.inputSize(1:2));
im_=im_ - net.meta.normalization.averageImage;
opts.batchNormalization = false ;
```

```
net.layers{end}.type = 'softmax';
res=vl_simplenn(net,im_);
scores=squeeze(gather(res(end).x));
[bestScore,best]=max(scores);
str=[subdir(best+2).name ':' num2str(bestScore)];
title(str);
disp(str);
```

从上面可以看出，就是载入模型，完成正确的预处理，然后进行分类。

一个样本的结果如下，0:0.99968，表示分类为类别 0 的概率是 0.99968，可知结果正确，0 代表的类别就是中性表情。



6 总结

有很多的优秀代码仍然使用 matconvnet，而且它的社区所包含的预训练模型也非常多，非常适合训练过程中进行调试，建议大家有 Matlab 环境和多余精力的可以学习一下，学习成本很低，技多不压身嘛。

【darknet 速成】Darknet 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

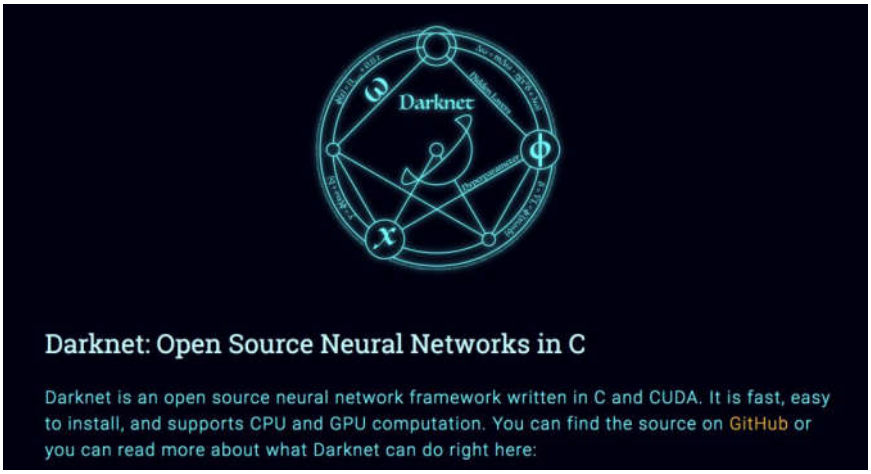
欢迎来到专栏《2 小时玩转开源框架系列》，这是我们第 12 篇文章，前面已经说过了 caffe，tensorflow，pytorch，mxnet，keras，paddlepaddle，cntk，chainer，deeplearning4j，matconvnet，lasagne。

本文说下 darknet，也是最后一个框架了，本文所用到的数据，代码请参考我们官方 git。

https://github.com/longpeng2008/yousan.ai

1 Darknet 是什么

首先不得不夸奖一下 Darknet 的主页风格不错。



官网地址：https://pjreddie.com/darknet/

GitHub： https://github.com/pjreddie/darknet

Darknet 本身是 Joseph Redmon 为了 Yolo 系列开发的框架。

Joseph Redmon，一个从 look once，到 look Better, Faster, Stronger，到 An Incremental Improvement，也就是从 Yolo v1，干到 Yolo v2，Yolo v3 的男人，头像很应景。

Darknet 几乎没有依赖库，是从 C 和 CUDA 开始撰写的深度学习开源框架，支持 CPU 和 GPU。

咱们的第一个开源框架说的是 Caffe，现在这最后一个 Darknet 跟 caffe 倒是颇有几分相似之处，只是更加轻量级。

2 Darknet 结构解读

首先我们看下 Darknet 的代码结构如下：

cfg	guys one of my beehives died :-(	6 months ago
data	SELU activation and yolo openimages	7 months ago
examples	GUYS I THINK MAYBE IT WAS BROKEN ON OPENCV IDK	6 months ago
include	GUYS I THINK MAYBE IT WAS BROKEN ON OPENCV IDK	6 months ago
python	faster nms and stuff	a year ago
scripts	coco dataset script	2 years ago
src	GUYS I THINK MAYBE IT WAS BROKEN ON OPENCV IDK	6 months ago

cfg，data，examples，include，python，src，scripts 几个子目录。

2.1 data 目录

labels	need the actual chars	2 years ago
9k.labels	adding yolo9000	2 years ago
9k.names	adding yolo9000	2 years ago
9k.tree	...and tree :tree:	2 years ago
coco.names	🔥 🔥 yolo v2 🔥 🔥	2 years ago
coco9k.map	adding yolo9000	2 years ago
dog.jpg	updates and things	3 years ago
eagle.jpg	NIPS	4 years ago
giraffe.jpg	giraffe	3 years ago
goal.txt	example	3 years ago
horses.jpg	NIPS	4 years ago
imagenet.labels.list	updates and things	3 years ago
imagenet.shortnames.list	updates and things	3 years ago
inet9k.map	adding yolo9000	2 years ago
kite.jpg	added tensorflow kite image to show how easy it is	a year ago
openimages.names	SELU activation and yolo openimages	7 months ago
person.jpg	NIPS	4 years ago
scream.jpg	New YOLO	3 years ago
voc.names	:psyduck:	2 years ago

以上就是 data 目录的内容，包含了各种各样的文件。图片就是测试文件了，不必说。我们首先看看 imagenet.labels.list 和 imagenet.shortnames.list 里面是什么。

imagenet.labels.list 是：

```
n02120505
n02104365
n02086079
n02101556
.....
```

看得出来就是 imagenet 的类别代号，与之对应的 imagenet.shortnames.list 里是：

```
kit fox
English setter
Siberian husky
Australian terrier
 .....
```

可知这两个文件配套存储了 imagenet1000 的类别信息。

接着看 9k.labels，9names，9k.trees，里面存储的就是 Yolo9000 论文中对应的 9418 个类别了。coco.names，openimages.names，voc.names 都类似。

2.2 cfg 目录

cfg，下面包含两类文件，一个是.data，一个是.cfg 文件。我们打开 imagenet1k.data 文件看下，可知它配置的就是训练数据集的信息：

```
classes=1000 ##分类类别数
train  = /data/imagenet/imagenet1k.train.list ##训练文件
valid  = /data/imagenet/imagenet1k.valid.list ##测试文件
backup = /home/pjreddie/backup/ ##训练结果保存文件夹
labels = data/imagenet.labels.list #标签
names  = data/imagenet.shortnames.list
top=5
```

另一类就是.cfg 文件，我们打开 cifar.cfg 文件查看。

```
##---------1 优化参数配置---------##
[net]
batch=128
subdivisions=1
height=28
width=28
channels=3
max_crop=32
min_crop=32

##数据增强参数
hue=.1
saturation=.75
exposure=.75

##学习率策略
learning_rate=0.4
policy=poly
power=4
```

```
max_batches = 5000 ##迭代次数
momentum=0.9 ##动量项
decay=0.0005 ##正则项
```

```
##---------2 网络参数配置---------##
[convolutional]
batch_normalize=1  ##是否使用 batch_normalization
filters=128
size=3
stride=1
pad=1
activation=leaky ##激活函数

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[dropout]
probability=.5

[convolutional]
batch_normalize=1
filters=256
size=3
```

```
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[dropout]
probability=.5

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
```

```
filters=512
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[dropout]
probability=.5

[convolutional]
filters=10
size=1
stride=1
pad=1
activation=leaky

[avgpool]

[softmax]
groups=1
```

包含两部分，第一部分就是优化参数的定义，类似于 caffe 的 solver.prototxt 文件。第二部分就是网络定义，类似于 caffe 的 train.prototxt 文件，不同的是网络层用[]来声明，batch normalization 以及激活函数等配置进了[convolutional]里面。

最后的 avgpool 不需要配置池化半径，softmax 不需要配置输入输出，在最后设置 group 参数。

你可能好奇，那残差网络怎么弄呢？

```
[shortcut]
activation=leaky
```

```
from=-3
```

如上，通过一个 from=-3 参数来进行配置，就是往后退 3 个 block 的意思了。

2.3 python 目录

下面只有两个文件，即 darknet.py 和 proverbot.py。前者就是 python 调用 yolo 模型的案例，后者没什么用。

2.4 include，src，examples 目录

include 和 src 就是具体的函数实现了，卷积等各类操作都在这里。examples 就是高层任务的定义，包括 classifier，detector，代码的解读就超过本文的内容了，以后详解。

3 数据准备和模型定义

3.1 数据准备

前面已经把该介绍的都介绍了，下面就开始准备数据进行训练。跟 caffe 一样，数据准备的流程非常简单。

首先，在 data 目录下建立我们自己的任务，按照如下目录，把文件准备好

```
├── genedata.sh
├── labels.txt
├── test
├── test.list
├── train
└── train.list
```

使用如下命令生成文件

```
find `pwd`/train -name \*.jpg > train.list
find `pwd`/test -name \*.jpg > test.list
```

其中每一行都存储一个文件，而标签是通过后缀获得的。

```
/Users/longpeng/Desktop/darknet/data/mouth/train/60_smile.jpg
/Users/longpeng/Desktop/darknet/data/mouth/train/201_smile.jpg
/Users/longpeng/Desktop/darknet/data/mouth/train/35_neutral.jpg
/Users/longpeng/Desktop/darknet/data/mouth/train/492_smile.jpg
```

标签的内容存在 labels.txt 里面，如下

```
neutral
smile
```

3.2 配置训练文件路径和网络

去 cfg 目录下建立文件 mouth.data 和 mouth.cfg，mouth.data 内容如下：

```
classes=2
train  = data/mouth/train.list
valid  = data/mouth/test.list
labels = data/mouth/labels.txt
backup = mouth/
top=5
```

mouth.cfg 内容如下:

```
[net]
batch=16
subdivisions=1
height=48
width=48
channels=3
max_crop=48
min_crop=48

hue=.1
saturation=.75
exposure=.75

learning_rate=0.01
policy=poly
power=4
max_batches = 5000
momentum=0.9
decay=0.0005

[convolutional]
batch_normalize=1
filters=12
size=3
stride=1
pad=1
activation=leaky
```

```
[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=24
size=1
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=48
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[connected]
output=128
activation=relu

[connected]
output=2
activation=linear
```

```
[softmax]
```

在这里我们用上了一点数据增强操作，大家在后面会看到它的威力。

4 模型训练

使用如下命令进行训练：

```
./darknet classifier train cfg/mouth.data cfg/mouth.cfg
```

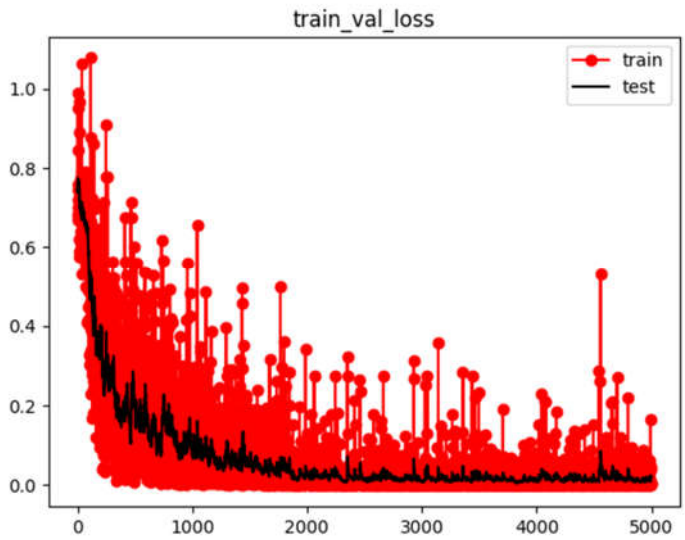
训练结果如下：

```
    0 conv     12  3 x 3 / 1    48 x  48 x   3   ->    48 x  48 x  12  0.001 BFLOPs
    1 max          2 x 2 / 2    48 x  48 x  12   ->    24 x  24 x  12
    2 conv     24  1 x 1 / 1    24 x  24 x  12   ->    24 x  24 x  24  0.000 BFLOPs
    3 max          2 x 2 / 2    24 x  24 x  24   ->    12 x  12 x  24
    4 conv     48  3 x 3 / 1    12 x  12 x  24   ->    12 x  12 x  48  0.003 BFLOPs
    5 max          2 x 2 / 2    12 x  12 x  48   ->     6 x   6 x  48
    6 connected                              1728  ->   128
    7 connected                               128  ->     2
    8 softmax                                        2
Saving weights to mouth//mouth_1.weights
Saving weights to mouth//mouth_2.weights
```

```
613, 10.898: 0.049500, 0.149785 avg, 0.005926 rate, 0.094553 seconds, 9808 images
Loaded: 0.000097 seconds
614, 10.916: 0.021373, 0.136944 avg, 0.005921 rate, 0.095087 seconds, 9824 images
Loaded: 0.000123 seconds
615, 10.933: 0.037522, 0.127002 avg, 0.005916 rate, 0.094510 seconds, 9840 images
```

上面每一行展示的分别是：batch数目，epoch数目，损失，平均损失，学习率，时间，见过的样本数目。

将最后的结果提取出来进行显示，损失变化如下，可知收敛非常完美。



训练完之后使用如下脚本进行测试。

```
./darknet classifier valid cfg/mouth.data cfg/mouth.cfg
mouth/mouth_50.weights
```

一个样本的结果如下：

```
darknet/data/mouth/test/27_smile.jpg, 1, 0.006881, 0.993119,
99: top 1: 0.960000, top 5: 1.000000
```

依次表示样本 darknet/data/mouth/test/27_smile.jpg，被分为类别 1，分类为 0 和 1 的概率是 0.006881，0.993119，该样本是第 99 个测试样本，此时 top1 和 top5 的平均准确率分为是 0.96 和 1。

到这里，我们只用了不到 500 个样本，就完成了一个精度不错的分类器的训练，如此轻量级的 darknet，我决定粉了。

5 总结

本文讲解了如何使用 darknet 深度学习框架完成一个分类任务，框架固然小众，但是速度真快，而且非常轻便，推荐每一个玩深度学习，尤其是计算机视觉的朋友都用起来。

【Lasagne 速成】Lasagne/Theano 图像分类从模型自定义到测试

作者 | 言有三（微信号 Longlongtogo）
编辑 | 言有三

欢迎来到专栏《2 小时玩转开源框架系列》，这是我们第 11 篇，前面已经说过了 caffe，tensorflow，pytorch，mxnet，keras，paddlepaddle，cntk，chainer，deeplearning4j，matconvnet。

今天说 Lasagne，本文所用到的数据，代码请参考我们官方 git

https://github.com/longpeng2008/yousan.ai

1 Lasagne 是什么

说了这么久的开源框架，我们好像一直忘了一个很老牌的框架，就是 theano 对不对，在 2008 年的时候，这个框架就由 Yoshua Bengio 领导的蒙特利尔 LISA 组开源了。

一直没说 theano 是因为它的使用成本真的有点高，需要从底层开始写代码构建模型，不过今天说的这个是封装了 theano 的高层框架，即 Lasagen，它使得 theano 使用起来更简单。

官网地址：http://lasagne.readthedocs.io/en/latest/index.html

GitHub： https://github.com/Lasagne/Lasagne

2 Lasagne 训练准备

2.1 Lasagne 安装

Lasagne 安装很简单，只需要在终端输入下面命令即可安装：

```
pip install Lasagne
```

2.2 数据读取

由于没有特别好的接口，因此我们自己定义一个类就行了，实现从数据集中读取，以及产生 list，格式就是每一个类存在一个单独的文件夹下，主体代码如下。

```
class Dataset:
    def __init__(self, rootpath, imgwidth, imgheight, trainratio=0.9):
        self.rootpath = rootpath
        list_dirs = os.walk(self.rootpath)
```

```
    count = 0
    numofclasses = 0
    self.subdirs = []

    ##遍历文件夹
    for root, dirs, files in list_dirs:
        for d in dirs:
            self.subdirs.append(os.path.join(root,d))
    label = 0
    self.imagedatas = []
    self.labeldatas = []
    for subdir in self.subdirs:
        images = glob.iglob(os.path.join(subdir,'*.jpg'))
        for image in images:
            imagedata = cv2.imread(image,1)
            imagedata = cv2.resize(imagedata,(imgwidth,imgheight))
            imagedata = imagedata.astype(np.float) / 255.0
            imagedata = imagedata - [0.5,0.5,0.5]
            imagedata = imagedata.transpose((2,0,1))

    self.imagedatas.append(imagedata)
    self.labeldatas.append(label)
    label = label + 1
    self.imagedatas = np.array(self.imagedatas).astype(np.float32)
    self.labeldatas = np.array(self.labeldatas).astype(np.int32)
    indices = np.arange(len(self.imagedatas))
    np.random.shuffle(indices)
    splitindex = int(trainratio*self.imagedatas.shape[0])

    self.imagetraindatas = self.imagedatas[0:splitindex].copy()
    self.labeltraindatas = self.labeldatas[0:splitindex].copy()
    self.imagevaldatas = self.imagedatas[splitindex:].copy()
    self.labelvaldatas = self.labeldatas[splitindex:].copy()

 ##定义数据迭代接口
def iterate_minibatches(self, inputs, targets, batchsize, shuffle=False):
    assert len(inputs) == len(targets)
```

```
    if shuffle:
        indices = np.arange(len(inputs))
        print "indices type=",type(indices)
        np.random.shuffle(indices)

    for start_idx in range(0, len(inputs) - batchsize + 1, batchsize):
        if shuffle:
            excerpt = indices[start_idx:start_idx + batchsize]
        else:
            excerpt = slice(start_idx, start_idx + batchsize)
        yield inputs[excerpt], targets[excerpt]
```

以上就实现了将一个数据集下的不同子文件夹的图片随机分成了训练集和测试集，并提供了统一的接口。当然这里只做了最简单的数据预处理而没有做数据增强，这就留待读者自己去完成了。

2.3 网络定义

基本上和所有 python 库的方法是一样的，调用接口就行。

```
def simpleconv3(input_var=None):
    network = lasagne.layers.InputLayer(shape=(None, 3, 48, 48),
                                        input_var=input_var)

    network = lasagne.layers.Conv2DLayer(
            network, num_filters=12, filter_size=(3, 3),
            nonlinearity=lasagne.nonlinearities.rectify,
            W=lasagne.init.GlorotUniform())
    network = batch_norm(network)
    network = lasagne.layers.MaxPool2DLayer(network, pool_size=(2, 2))

    network = lasagne.layers.Conv2DLayer(
            network, num_filters=24, filter_size=(3, 3),
            nonlinearity=lasagne.nonlinearities.rectify,
            W=lasagne.init.GlorotUniform())
    network = batch_norm(network)
    network = lasagne.layers.MaxPool2DLayer(network, pool_size=(2, 2))
    network = lasagne.layers.Conv2DLayer(
            network, num_filters=48, filter_size=(3, 3),
```

```
        nonlinearity=lasagne.nonlinearities.rectify,
        W=lasagne.init.GlorotUniform())
    network = batch_norm(network)
    network = lasagne.layers.MaxPool2DLayer(network, pool_size=(2, 2))

    network = lasagne.layers.DenseLayer(
        lasagne.layers.dropout(network, p=.5),
        num_units=128,
        nonlinearity=lasagne.nonlinearities.rectify)

    network = lasagne.layers.DenseLayer(
        lasagne.layers.dropout(network, p=.5),
        num_units=2,
        nonlinearity=lasagne.nonlinearities.softmax)

    return network
```

以上定义的就是一个 3 层卷积 2 层全连接的网络，使用 lasagne.layers 接口。

3 模型训练

1、首先通过 Theano 里的 tensor 对输入和输出进行定义

```
input_var = T.tensor4('inputs')
target_var = T.ivector('targets')
```

inputs 是一个四维的张量，targets 是一个 ivector 变量。

2、调用 lasagne.objectives 里的损失函数接口：

```
network = simpleconv3(input_var)
prediction = lasagne.layers.get_output(network)
loss =lasagne.objectives.categorical_crossentropy(prediction, target_var)
loss = loss.mean()
```

network 即网络模型，prediction 表示它的输出，损失函数 categorical_crossentropy 就是交叉熵了。

验证集和测试集上的定义与此类似，只需要更改 deterministic 为 deterministic=True，这样会屏蔽掉所有的 dropout 层，如下：

```
test_prediction = lasagne.layers.get_output(network, deterministic=True)
test_loss = lasagne.objectives.categorical_crossentropy(test_prediction,
target_var)
```

```
test_loss = test_loss.mean()
```

接下来就是训练方法，使用 nesterov_momentum 法：

```
params = lasagne.layers.get_all_params(network, trainable=True)
updates = lasagne.updates.nesterov_momentum( loss, params,
learning_rate=0.01, momentum=0.9
)
```

最后定义训练函数：

```
train_fn = theano.function([input_var, target_var], loss, updates=updates)
```

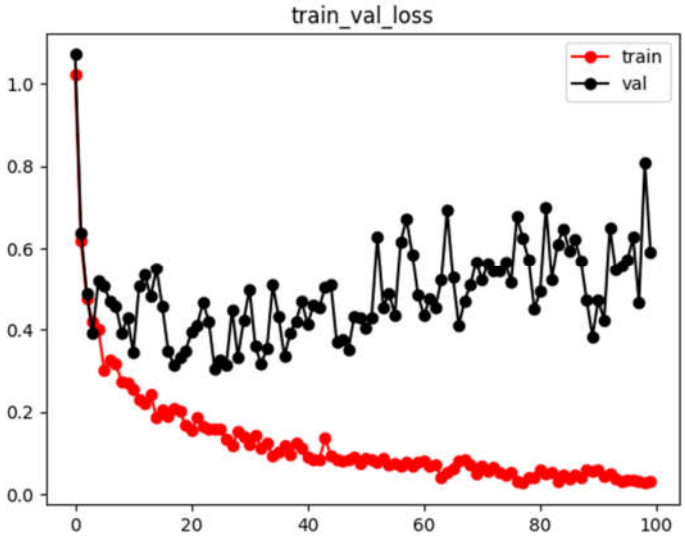
接收两个输入 input_var， target_var，利用 updates 表达式更新参数。如果是用于验证和测试，就不需要进行网络参数的更新，而且可以增加精度等变量，这时这样定义：

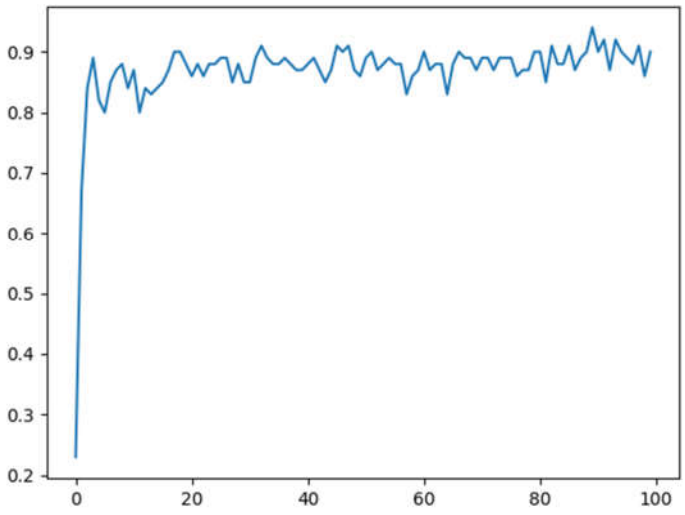
```
test_acc = T.mean(T.eq(T.argmax(test_prediction, axis=1), target_var),
dtype=theano.config.floatX
)
val_fn = theano.function([input_var, target_var], [test_loss, test_acc])
```

最后，每一个 epoch 取得数据训练进行训练：

```
train_data =
mydataset.iterate_minibatches(mydataset.imagetraindatas,mydataset.labe
ltraindatas,16,True)
train_batches = 0
  for input_batch, target_batch in train_data:
     train_loss += train_fn(input_batch, target_batch)
     prediction = lasagne.layers.get_output(network)
     train_batches += 1
  print("Epoch %d: Train Loss %g" % (epoch + 1, train_loss /
train_batches))
```

结果如下，老样子，测试集合精度 90%，模型过拟合。





以上就是 Lasagne 从数据准备，模型定义到输出结果的整个流程，想要体验可以去参考 git 代码。

4 总结

Lasagne/Theano 给我最大的感觉就是慢，比至今用过的每一个框架都要慢，不过了解一下并没有坏处，毕竟 Theano 曾经辉煌。

【移动端 DL 框架】当前主流的移动端深度学习框架一览

大家好，继之前的 12 大深度学习开源框架之后，我们准备开通新的专栏《移动端 DL 框架》，这是第一篇文章，先来做一个总体的介绍，更多的细节可以关注以后的文章。

在这个专栏中，我们会介绍与移动端的模型训练和部署有关的框架的使用。

作者&编辑 | 言有三

深度学习模型要落地，比如要部署到手机等移动端平台，之前给大家介绍的用于训练的框架就不能直接使用了，我们需要相应的移动端框架，目前国内外各大公司纷纷开源自家的框架。

1 TensorFlow Lite

这是 Google 在 2017 年 I/O 开发者大会上开源的将 TensorFlow 训练好的模型迁移到 Android App 的框架，地址和一些学习资源如下：

TensorFlow Lite 使用 Android Neural Networks API，默认调用 CPU，目前最新的版本已经支持 GPU。

项目地址和相关学习资源如下。

https://tensorflow.google.cn/lite/

https://github.com/amitshekhariitbhu/Android-TensorFlow-Lite-Example

2 Core ML

Core ML 是 2017 年 Apple 公司在 WWDC 上与 iOS11 同时发布的移动端机器学习框架，底层使用 Accelerate 和 Metal 分别调用 CPU 和 GPU。Core ML 需要将你训练好的模型转化为 Core ML model，它的使用流程如下：

在一年之后，也就是 2018 年 WWDC 上，Apple 发布了 Core ML 2，主要改进就是通过权重量化等技术优化模型的大小，使用新的 Batch Predict API 提高模型的预测速度，以及容许开发人员使用 MLCustomLayer 定制自己的 Core ML 模型。

项目地址和相关学习资料如下：

https://developer.apple.com/documentation/coreml

https://github.com/likedan/Awesome-CoreML-Models

3 Caffe2

Caffe2 是 facebook 在 2017 年发布的一个跨平台的框架，不仅仅支持 Windows，Linux，Macos 三大桌面系统，也支持移动端 iOS，Android，可以说是集训练和推理于一身。

Caffe2 本来就是基于 caffe 开发的，Caffe 基于 C++开发，所以可以很自然地移植到移动端，目前 Caffe2 已经全部并入 Pytorch。两者的区别就是 PyTorch 是为研究而开发，更加灵活。Caffe2 是专为移动生产环境而开发，更加高效。

项目地址以及其相关的 model zoo 地址如下。

https://github.com/facebookarchive/caffe2

https://caffe2.ai/docs/zoo.html

https://github.com/caffe2/models

4 NCNN

ncnn 是 2017 年腾讯优图实验室开源的移动端框架，使用 C++ 实现，支持 Android 和 IOS 两大平台。

ncnn 已经被用于腾讯生态中的多款产品，包括微信，天天 P 图等。

项目地址和相关学习资料如下。

https://github.com/Tencent/ncnn

https://github.com/BUG1989/caffe-int8-convert-tools.git

5 Paddle-Mobile

Paddle-Mobile 是 2017 年百度 PaddlePaddle 组织下的移动端深度学习开源框架，当时叫做 mobile-deep-learning(MDL)。支持安卓和 ios 平台，CPU 和 GPU 使用，提供量化工具。

可以直接使用 Paddle Fluid 训练好的模型，也可以将 Caffe 模型进行转化，或者使用 ONNX 格式的模型。

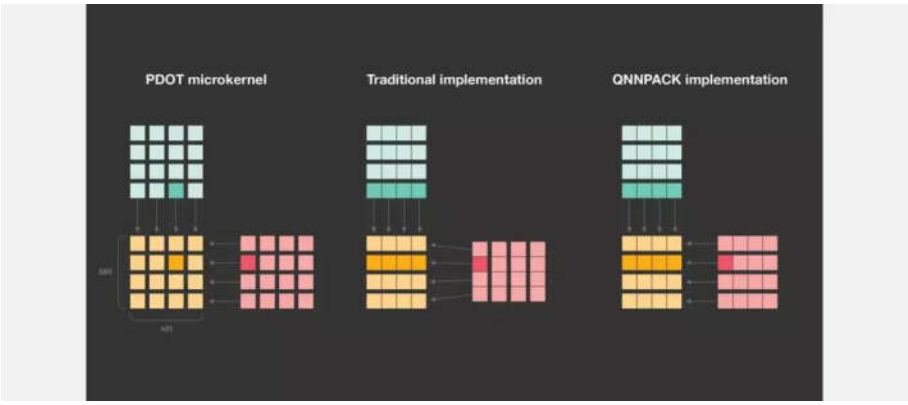
项目地址如下：

https://github.com/PaddlePaddle/paddle-mobile

https://github.com/PaddlePaddle/Paddle

6 QNNPACK

QNNPACK 是 Facebook 在 2018 年发布的 int8 量化低精度高性能开源框架，全称 Quantized Neural Network PACKage，用于手机端神经网络计算的加速，已经被整合到 PyTorch 1.0 中，在 Caffe2 里就能直接使用。



这个框架可以为很多运算加速，比如 DW 卷积 (Depthwise Convolution) ，目前支持的列表如下：

- [x] 2D Convolution
- [x] 2D Deconvolution
- [x] Channel Shuffle
- [x] Fully Connected
- [ ] Locally Connected
- [x] 2D Max Pooling
- [x] 2D Average Pooling
- [x] Global Average Pooling
- [x] Sigmoid
- [x] Leaky ReLU
- [x] Clamp (can be used for ReLU, ReLU6 if it is not fused in another operator)
- [x] SoftArgMax (aka SoftMax)
- [ ] Group Normalization

项目地址如下。

https://github.com/pytorch/QNNPACK

7 MACE

MACE 是 2018 年小米在开源中国开源世界高峰论坛中宣布开源的移动端框架，以 OpenCL 和汇编作为底层算子，提供了异构加速可以方便在不同的硬件上运行模型，同时支持各种框架的模型转换。

项目地址和相关学习资源如下：

https://github.com/XiaoMi/mace

https://github.com/XiaoMi/mace-models

8 MNN

MNN 是 2019 年阿里开源的移动端框架，不依赖第三方计算库，使用汇编实现核心运算，支持 Tensorflow、Caffe、ONNX 等主流模型文件格式，支持 CNN、RNN、GAN 等常用网络。作为后起之秀，自然是吸取了前面开源的这些移动端推理框架的所有优点。

已经用于阿里的淘宝，优酷等多个应用，覆盖短视频、搜索推荐等场景。

项目地址和学习资源如下：

https://github.com/alibaba/MNN

9 其他

除了上面这些正式发布的开源框架，还有一些其他的资源。比如不开源的骁龙的官方 SDK SNPE，主要支持自家的 DSP、GPU 和 CPU。

还有很早就存在的 GitHub 项目 caffe-android-lib，用于将 Caffe 往移动端进行移植，其实各家 AILab 应该自己都会有一套这样的工具。

以及 caffe 量化工具包 caffe-int8-convert-tools 等。

https://github.com/sh1r0/caffe-android-lib

https://github.com/BUG1989/caffe-int8-convert-tools

总结

这一次先让大家对移动端的深度学习框架有一个印象，后面我们会一个一个进行学习，敬请期待。

【杂谈】一招，同时可视化 18 个开源框架的网络模型结构和权重

深度学习开源框架众多，对于开发者来说其中有一个很硬的需求，就是**模型结构和权重的可视化。**使用过 Caffe 的同学都因为强大的 Netscope 可以离线修改实时可视化网络结构而暗爽，那其他的框架怎么样呢?

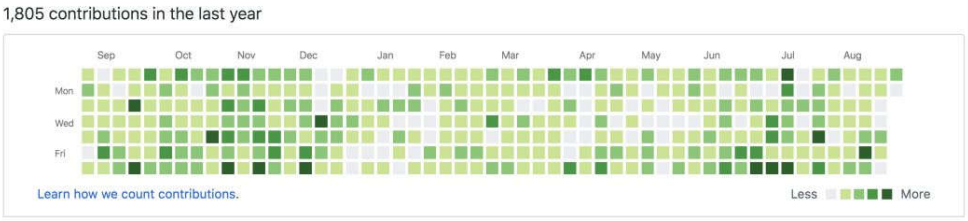
今天给大家介绍一个可以离线可视化各大深度学习开源框架模型结构和权重的项目，**netron。**

作者&编辑 | 言有三

1 项目介绍

项目开发者 Lutz Roeder，一位来自于微软 Visual Studio 团队的小哥，按照他自己的介绍，就是在家搞点 AI tools 玩玩。

这是过去一年的 contributions，基本就没有停过，这是真正硬核的开源贡献者呀，个人主页 https://www.lutzroeder.com/ai，有兴趣可以去瞧瞧。



Netron 是他开源的深度学习模型可视化工具，项目地址为：

NETRON

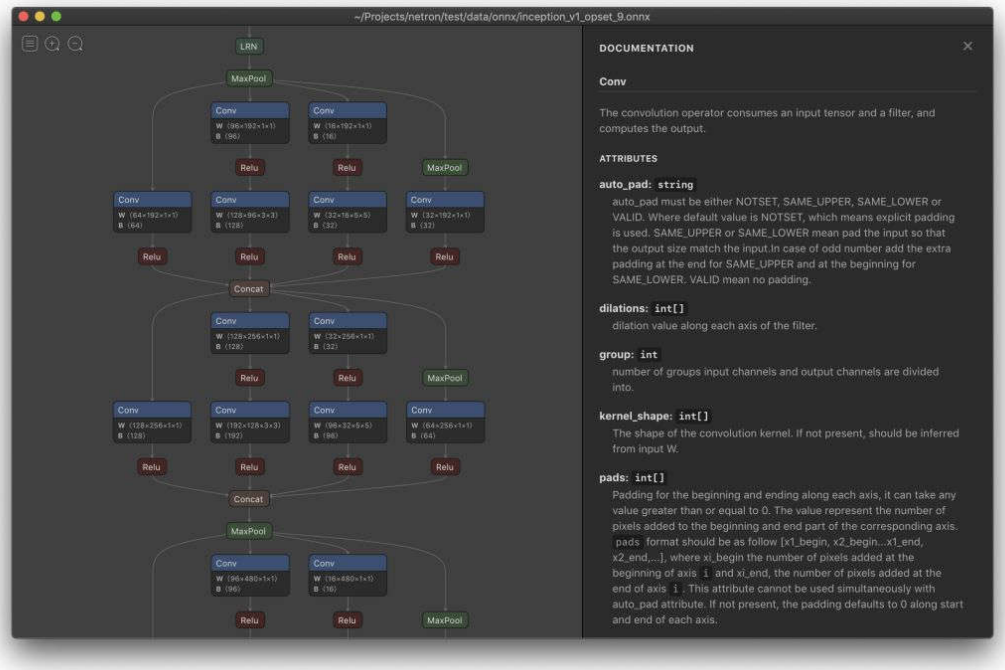
https://github.com/lutzroeder/netron

目前支持哪些框架呢？

- ONNX (.onnx, .pb, .pbtxt)
- Keras (.h5, .keras)
- Core ML (.mlmodel)
- Caffe (.caffemodel, .prototxt)
- Caffe2 (predict_net.pb, predict_net.pbtxt)
- MXNet (.model, -symbol.json)
- TorchScript (.pt, .pth)
- NCNN (.param)
- TensorFlow Lite (.tflite)
- PyTorch (.pt, .pth)
- Torch (.t7)
- CNTK (.model, .cntk)
- Deeplearning4j(.zip)
- PaddlePaddle (.zip, __model__)
- Darknet (.cfg)
- scikit-learn (.pkl)

- TensorFlow.js (model.json, .pb)
- TensorFlow (.pb, .meta, .pbtxt).

共 18 个框架，除了 chainer，matconvnet 等框架基本上把数得上名字的框架一网打尽，下面是 ONNX 的可视化界面，很 visual studio 的感觉。



另一方面，小哥哥也是非常的贴心，提供了各大平台的安装包！**macOS 的.dmg，Linux 的.deb，Windows 的.exe，还有浏览器版本，Python 服务器版本，真是 good man，小白们再也不用担心环境配置问题。**

2 可视化实验

下面我们就来尝试几个框架的可视化结果，首先要祭出有三 AI 开源的 **12 大深度学习开源框架的项目**，从模型和数据接口定义，到训练测试可视化，提供了全套代码，地址如下：

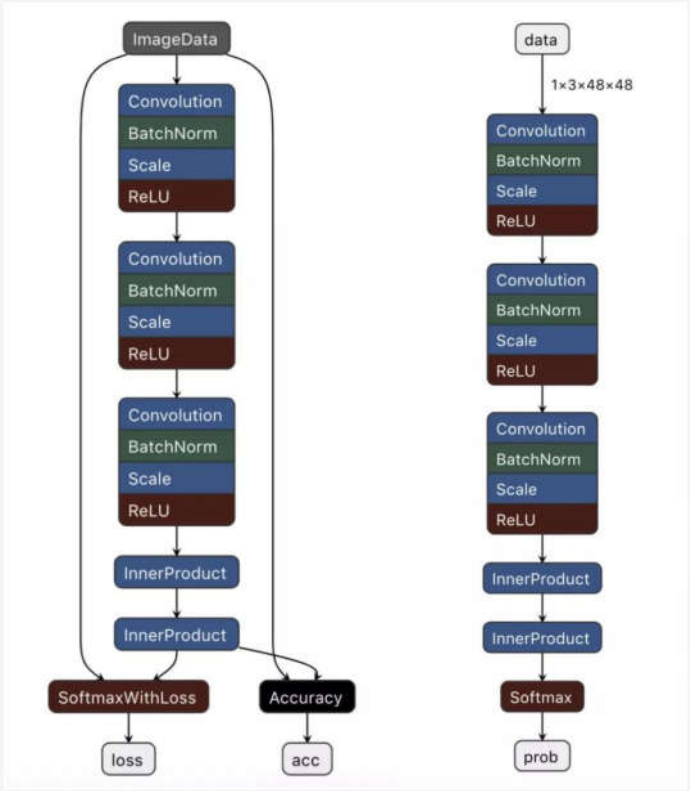
https://github.com/longpeng2008/yousan.ai

具体的教程大家可以跳转阅读原文。

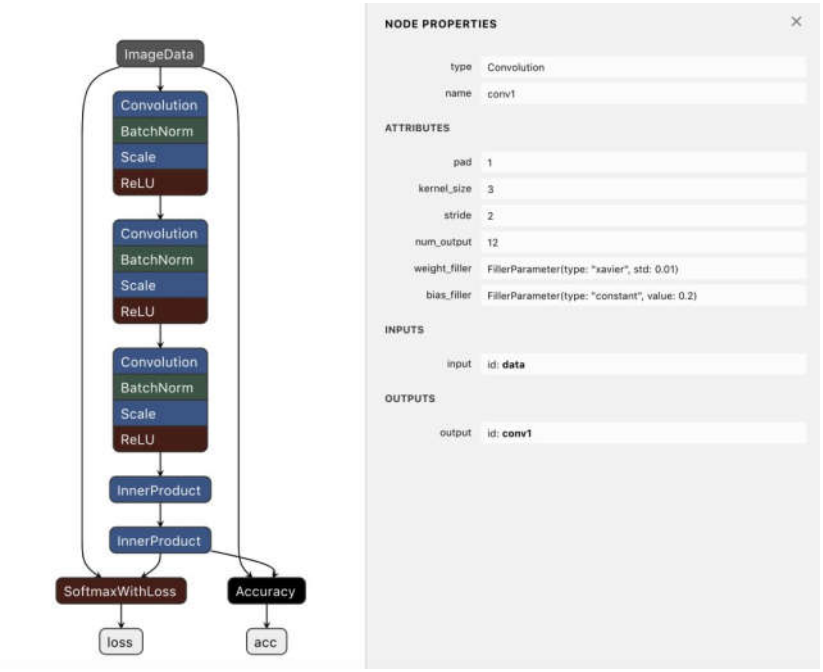
这 12 个框架我们都使用一个 **3 层的卷积模型**来完成一个图像分类问题，下面挑选其中几个常用的来体验可视化效果。

2.1 Caffe

Caffe 模型可视化的输入可以是 prototxt 文件和 caffemodel 文件。下面首先分别可视化训练网络和测试网络 train.prototxt 和 deploy.prototxt，结果如下：

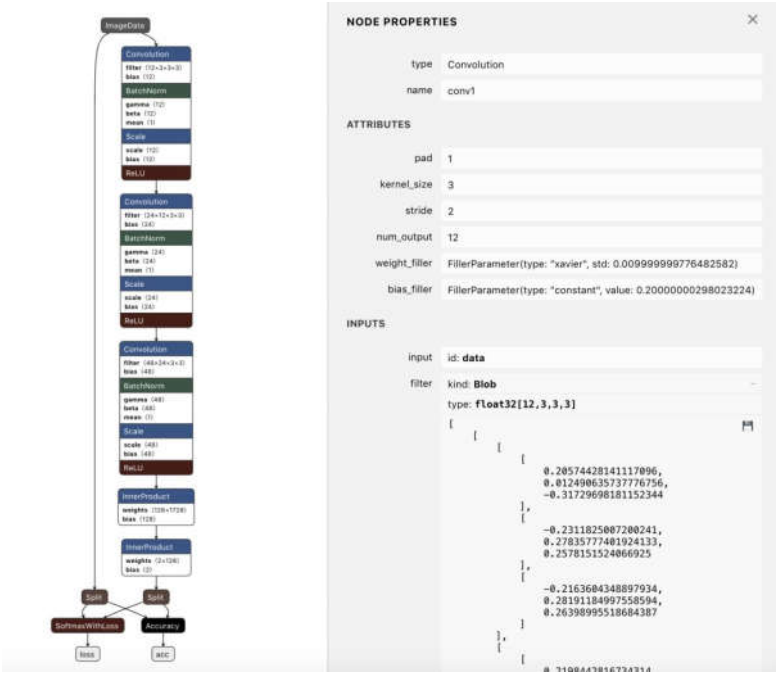


比起 Netscope，是不是效果也不遑多让。如果想要查看某一个网络层的细节，就可以点击该网络层，结果如下，所有网络配置参数细节皆可见，完美！



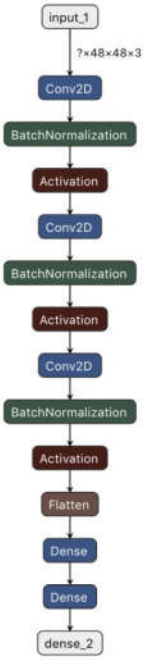
这就是全部了吗？当然不，你还可以直接载入.caffemodel 权重文件，直接查看每一个网络层的权重！

老司机们可以从中**简单统计权重的分布**，还可以一键导出参数为 npy 文件，看到那个**保存小按钮**没有，这就是细节，不得不再次给小哥哥点赞

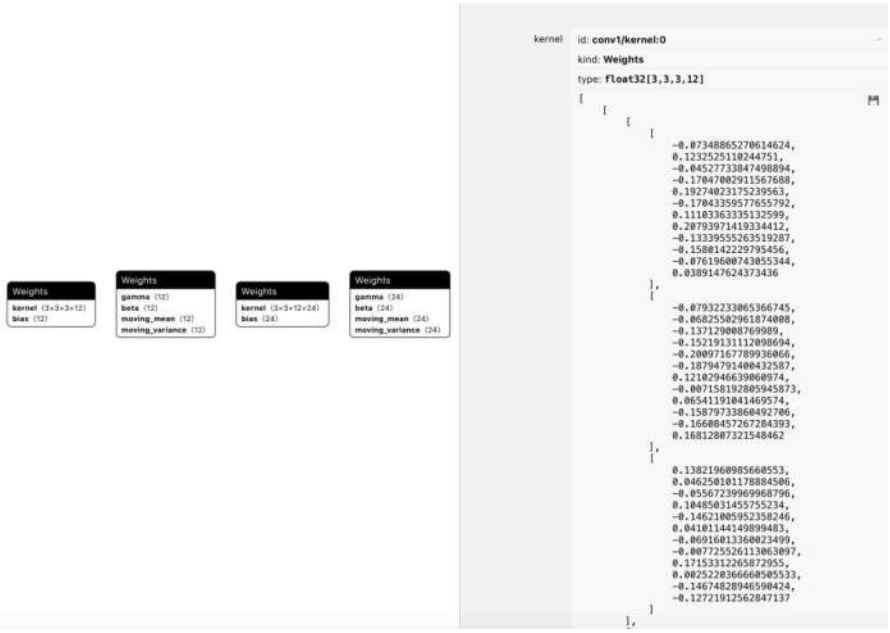


2.2 keras

keras 的可视化输入是 json 格式的模型文件，可以通过 model.to_json()将模型存储下来，然后载入.json 文件。

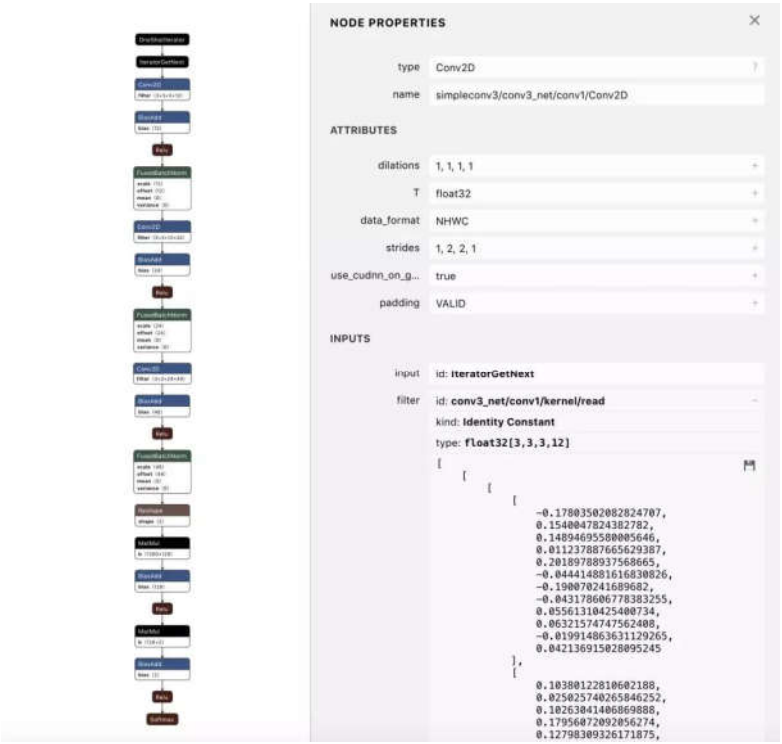


如果想查看权重，就载入.h5 文件，结果如下，虽然没有 caffe 的那么漂亮，也是很直观的，不过权重参数矩阵顺序不太一样。



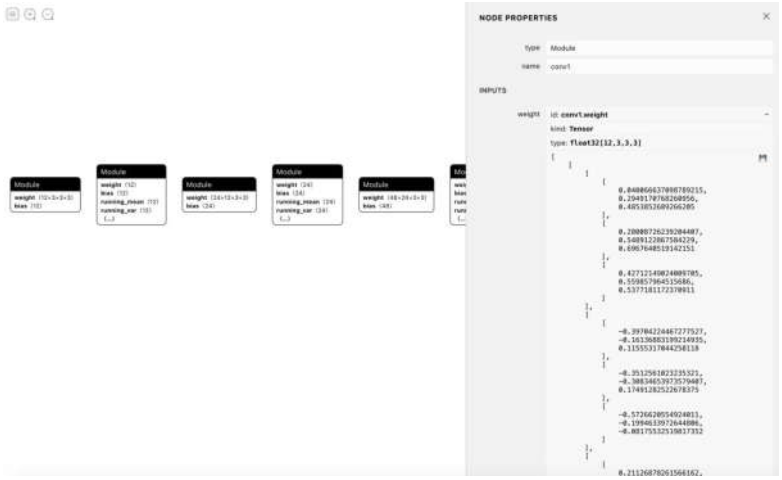
2.3 tensorflow

要想可视化 tensorflow 的模型结构，就必须将模型存储为 pb 格式，这样就能同时保存网络结构和参数了，结果如下。



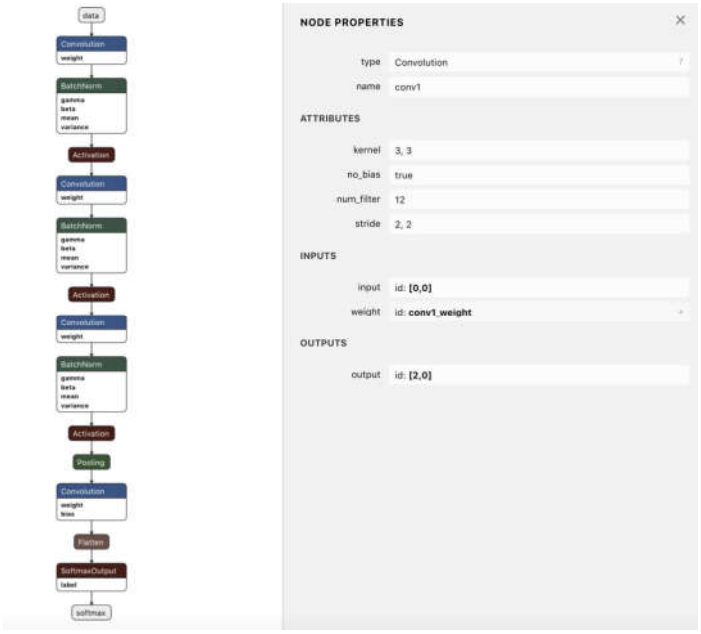
2.4 pytorch

pytorch 的网络结构可视化不支持，不过可以像 keras 一样查看 pt 权重文件。如果想可视化网络结构，可以使用 yousan.ai 项目中 pytorch 目录下的 visualize.py 脚本。



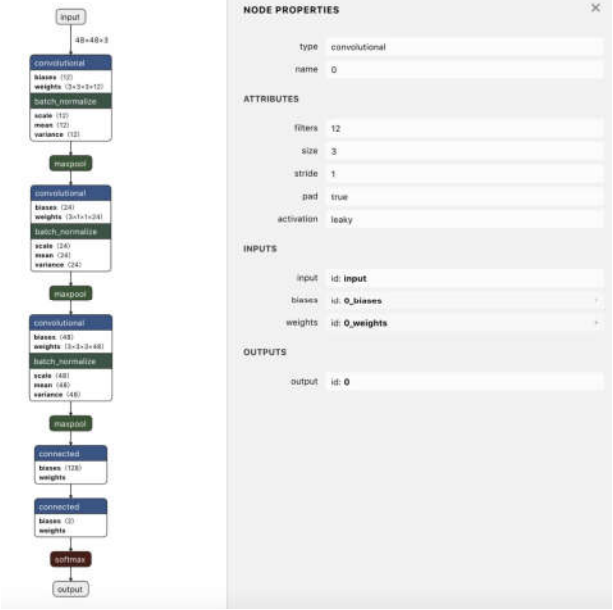
2.5 Mxnet

Mxnet 通过 symbol 接口定义网络，网络结构一般存在后缀为 symbol.json 的文件中，因此载入该文件即可进行可视化。权重的可视化原理类似，就不做赘述。

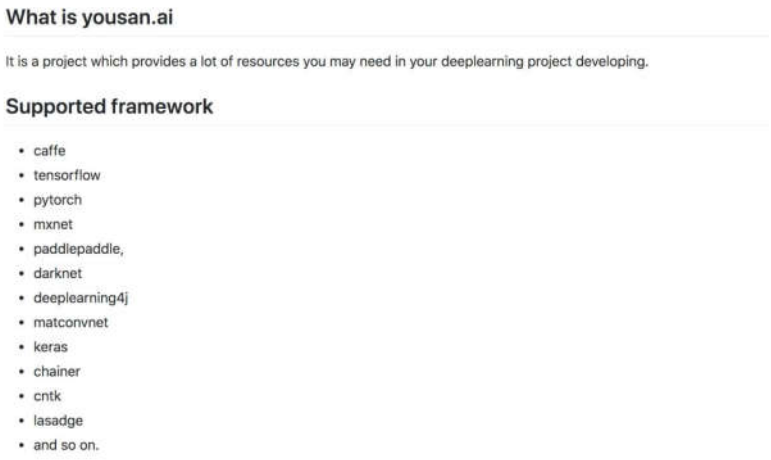


2.6 Darknet

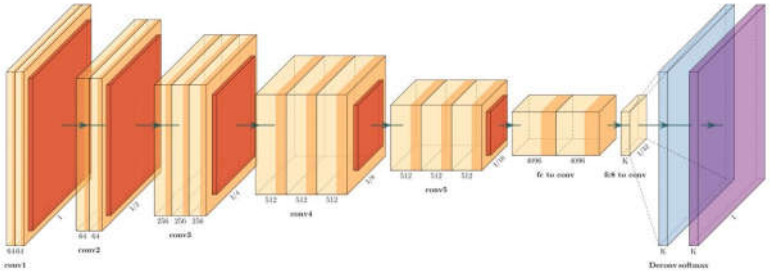
DarkNet 的网络结构定义在 cfg 文件中，载入该 cfg 文件即可进行可视化。



其他开源框架的案例，大家可以去下载我们的开源框架项目进行尝试，感谢小哥作出的贡献！



另外再安利一下我们之前的讲述如何绘制更好看的网络结构的文章，一起享用，保证更香。



总结

平时多留意一些好的工具，可以大大提升我们的学习效率，养成好的学习习惯，欢迎大家给我们推荐好的工具，在有三 AI 知识星球社区也可以一起分享。

【杂谈】那些酷炫的深度学习网络图怎么画出来的？

本文我们聊聊如何才能画出炫酷高大上的神经网络图，下面是常用的几种工具。

作者&编辑 | 言有三

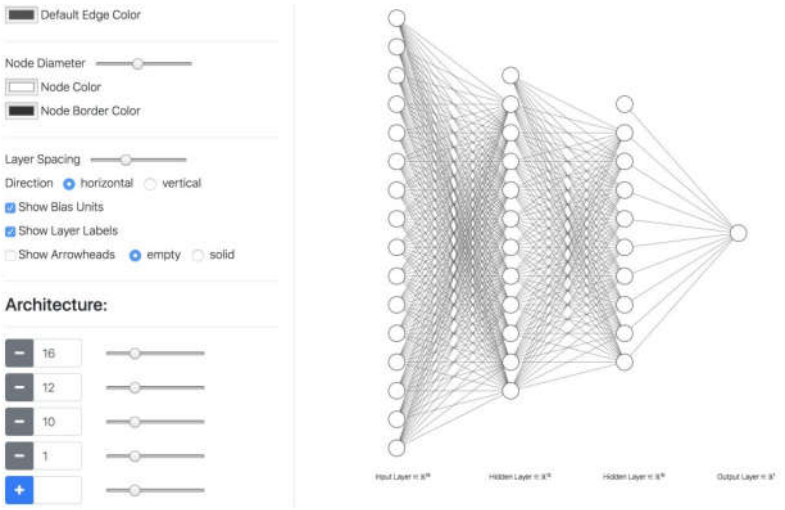
1 NN-SVG

这个工具可以非常方便的画出各种类型的图，是下面这位小哥哥开发的，来自于麻省理工学院弗兰克尔生物工程实验室， 该实验室开发可视化和机器学习工具用于分析生物数据。

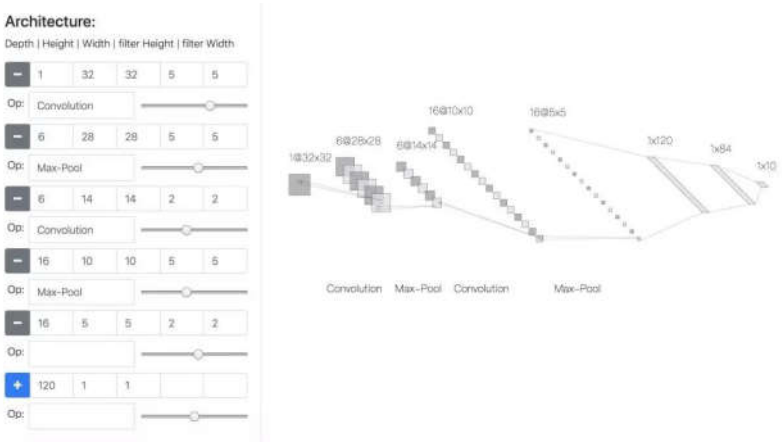
github 地址：*https://github.com/zfrenchee*

画图工具体验地址：http://alexlenail.me/NN-SVG/

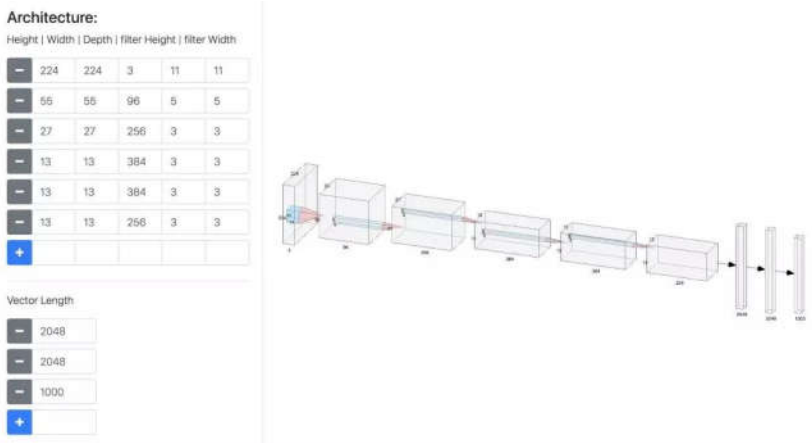
可以绘制的图包括以节点形式展示的 FCNN style，这个特别适合传统的全连接神经网络的绘制。



以平铺网络结构展示的 LeNet style，用二维的方式，适合查看每一层 featuremap 的大小和通道数目。



以三维 block 形式展现的 AlexNet style，可以更加真实地展示卷积过程中高维数据的尺度的变化，目前只支持卷积层和全连接层。



这个工具可以导出非常高清的 SVG 图，值得体验。

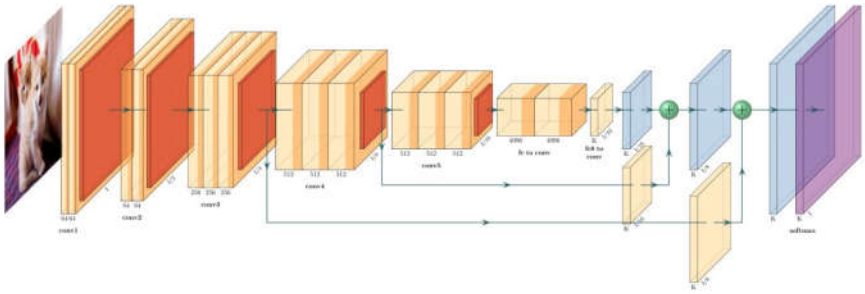
2 PlotNeuralNet

这个工具是萨尔大学计算机科学专业的一个学生开发的，一看就像计算机学院的嘛。

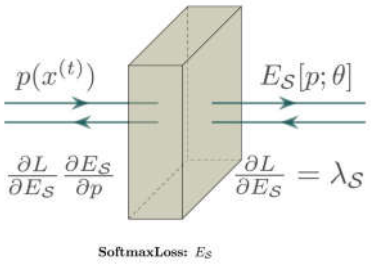
首先我们看看效果，其 github 链接如下，将近 4000 star：

https://github.com/HarisIqbal88/PlotNeuralNet

看看人家这个 fcn-8 的可视化图，颜值奇高。



使用的门槛相对来说就高一些了，用 LaTex 语言编辑，所以可以发挥的空间就大了，你看下面这个 softmax 层，这就是会写代码的优势了。



其中的一部分代码是这样的，会写吗。

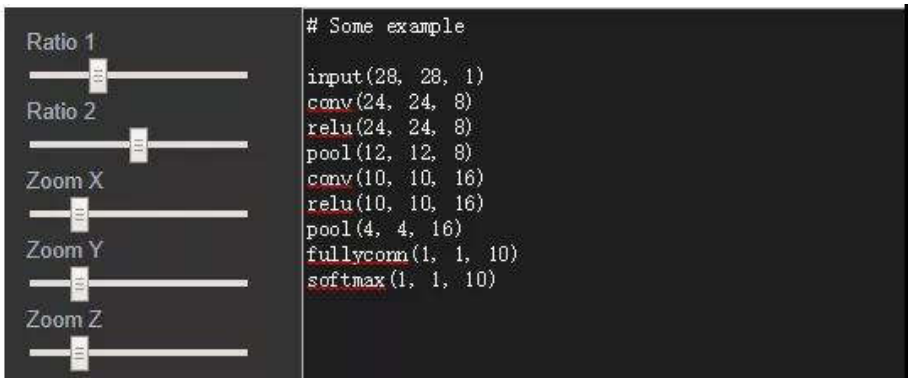
```
\pic[shift={(0,0,0)}] at (0,0,0) {Box={name=crp1,caption=SoftmaxLoss: $E_\mathcal{S}$ ,%
fill={rgb:blue,1.5;red,3.5;green,3.5;white,5},opacity=0.5,height=20,width=7,depth=20}};
```

相似的工具还有：*https://github.com/jettan/tikz_cnn*

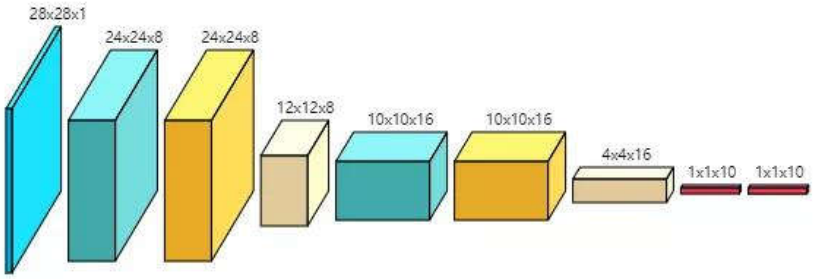
3 ConvNetDraw

ConvNetDraw 是一个使用配置命令的 CNN 神经网络画图工具，开发者是香港的一位程序员，Cédric cbovar。

采用如下的语法直接配置网络，可以简单调整 x，y，z 等 3 个维度，github 链接如下：*https://cbovar.github.io/ConvNetDraw/*



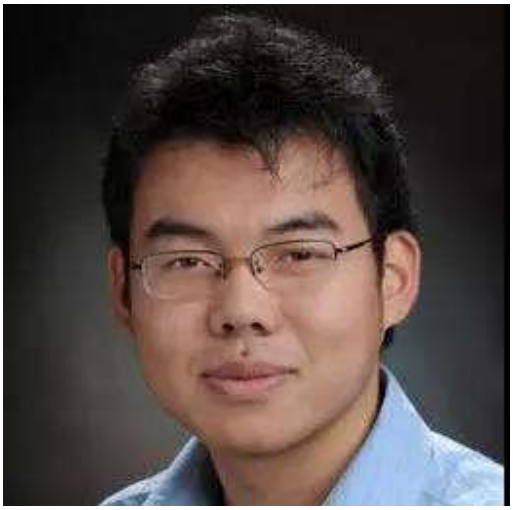
使用方法如上图所示，只需输入模型结构中各层的参数配置。



挺好用的不过它目标分辨率太低了，放大之后不清晰，达不到印刷的需求。

4 Draw_Convnet

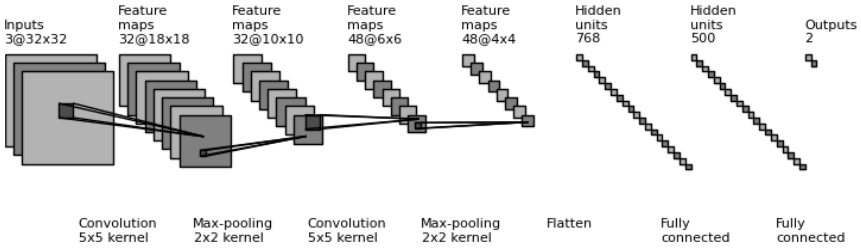
这一个工具名叫 draw_convnet，由 Borealis 公司的员工 Gavin Weiguang Ding 提供。



简单直接，是纯用 python 代码画图的，

https://github.com/gwding/draw_convnet

看看画的图如下，核心工具是 matplotlib，图不酷炫，但是好在规规矩矩，可以严格控制，论文用挺合适的。

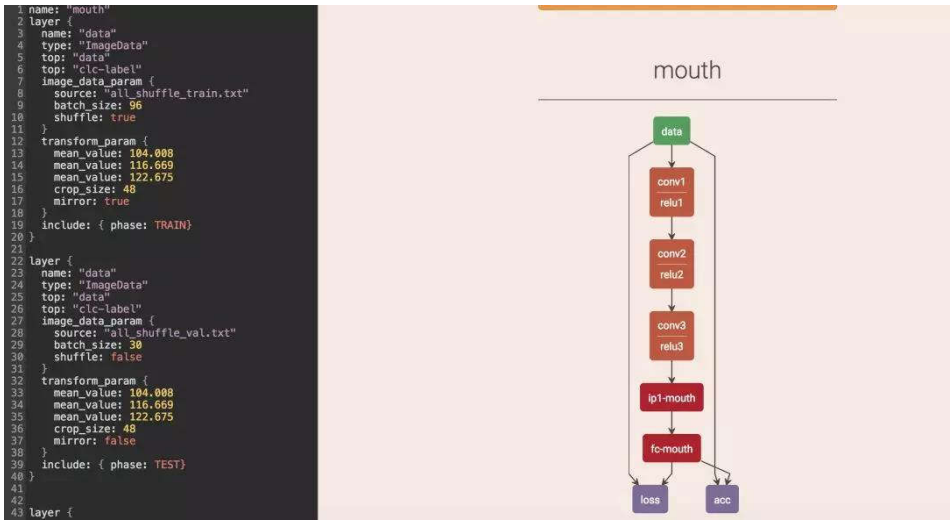


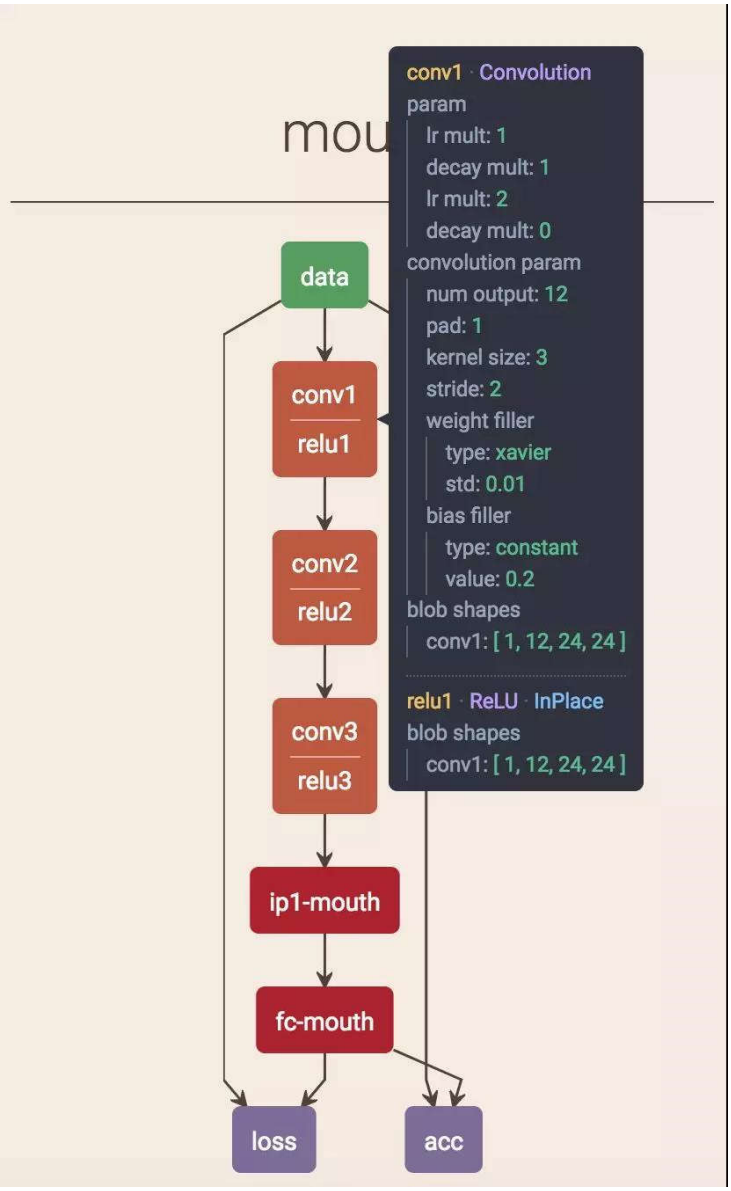
类似的工具还有：*https://github.com/yu4u/convnet-drawer*

5 Netscope

下面要说的是这个，我最常用的，caffe 的网络结构可视化工具，大名鼎鼎的 netscope，由斯坦福 AILab 的 Saumitro Dasgupta 开发，找不到照片就不放了，地址如下：

https://github.com/ethereon/netscope





左边放配置文件，右边出图，非常方便进行网络参数的调整和可视化。这种方式好就好在各个网络层之间的连接非常的方便。

其他

再分享一个有意思的，不是画什么正经图，但是把权重都画出来了。

http://scs.ryerson.ca/~aharley/vis/conv/

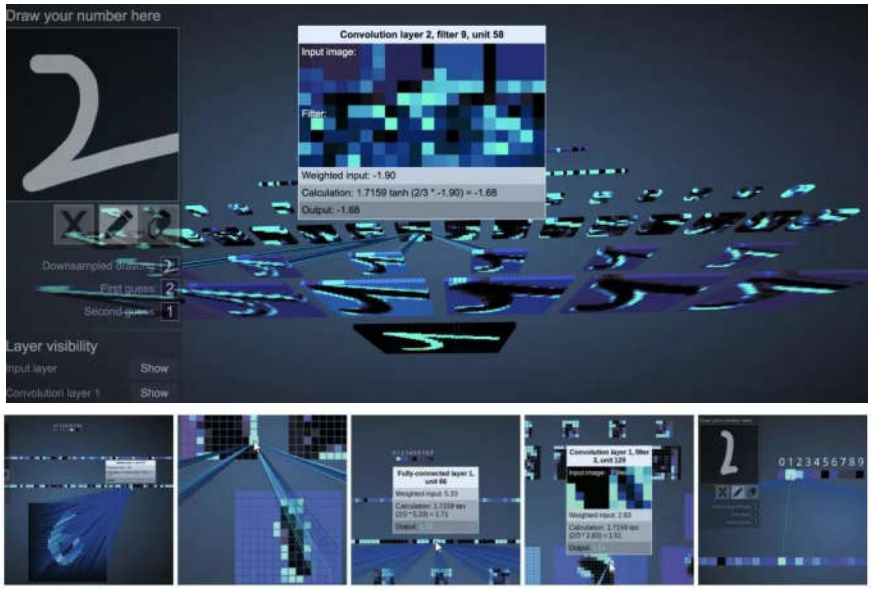
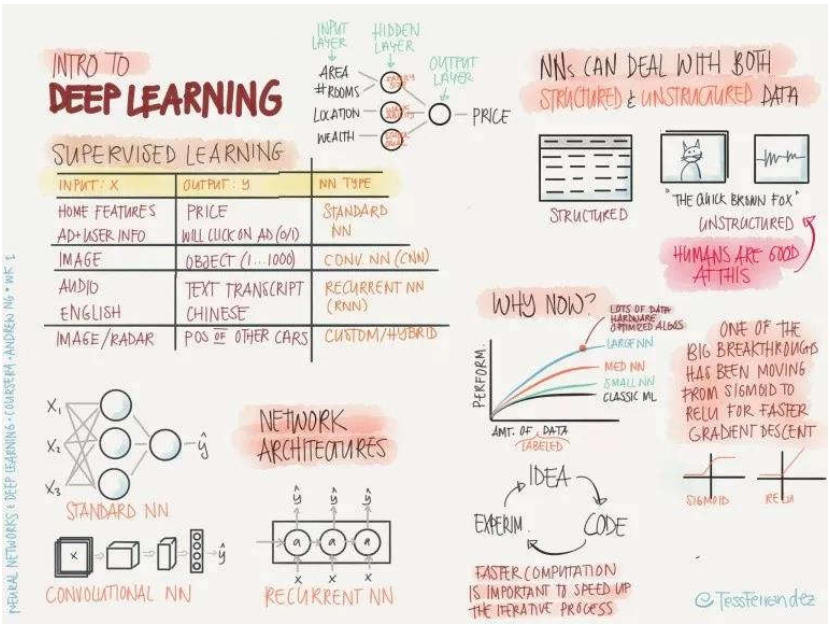


Fig. 4. Screenshots from the visualization. From left to right, the images show: edges revealed by hovering on a fully-connected node, edges revealed by hovering on a convolutional node, details revealed by clicking on a fully-connected node, details revealed by clicking on a convolutional node, and finally the top layer activations of a network given ambiguous input. This figure is best viewed in a zoomable PDF.

看了这么多，有人已经在偷偷笑了，上 PPT 呀，想要什么有什么，想怎么画就怎么画。不过妹子呢？怎么不来开发一个粉色系的可视化工具呢？类似于这样的



第一次插个广告？

总结

那么，你都用什么画呢？欢迎留言分享一下！

三人行必有 AI，欢迎关注有三 AI

未完待续

有三 AI，2020 年 6 月 20 日