机器学习编程作业 3-例程

By goodluckcwl Email: weiliangchen@zju.edu.cn

人脸识别理论介绍

人脸识别可以分成**识别（identification）**跟**验证(verification)**两种类型。其中，前者是指把一张人脸识别为一个具体的人，后者是指**给定两张人脸判断其是否是属于同一个人。**

对于人脸识别测试来说，根据场景的不同可分为两种类型。1) 测试集中所有的 identities 都在训练集中已预先定义。在这种情况下，人脸识别（face identification）就是一个分类问题。2）测试集中的 identities 在训练集中不存在，此时人脸识别（face identification）可以看成人脸验证（face verification）。

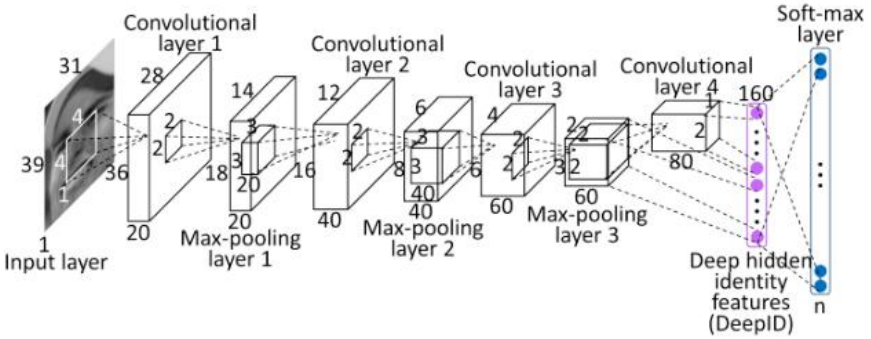
本次编程作业的要求是做**人脸验证（ face verification）**，即测试集中的人在训练集中并不存在。在这种情况下，思路是把人脸图像映射到一个特征空间，使得同一个人的图片距离更近，不同人的图片距离更远，这样就可以根据两张图片的特征向量之间的距离来判断是否是属于同一个人。如图：

因此，目前人脸验证的大致思路可分为两步：

- 学习一个特征空间
- 把图片映射到特征空间并且根据距离进行判决

学习特征空间

在 2014 年的时候，DeepID[1]提出了通过大规模的人脸分类任务来学习特征空间的方法。如图，输入图像经过卷积、池化等非线性映射，最终映射成 160 维的特征。DeepID 采用了交叉熵损失作为监督，其想法是：训练一个分类的网络，当分类的类别数很多的时候，这个分类网络的特征可以用来做验证。通过训练分类网络学习到了一个有效的人脸的表达。



在 2016 年的时候，Wen Y 等[2]提出判别式特征学习方法，即在训练分类网络的时候不仅仅要能够分类，还要使得类内更加聚集。在 2017 年的时候，Liu W 等[3]提出 Sphereface，

要求训练分类网络的时候类间的最小距离要大于类内的最大距离。沿着此思路，此后又有多种不同的改进[4,5,6]。

另一种学习特征空间的思路是度量学习。2015 年，谷歌提出了 Triplet Loss 损失函数[7]，每次选择一个 Anchor,然后再随机选取一个和 Anchor 属于同一类的正样本和不同类的负样本。训练的目标就是使 Anchor 和正样本距离尽可能小，Anchor 和负样本距离尽可能大。

参考文献

[1].Sun Y, Wang X, Tang X. Deep Learning Face Representation from Predicting 10,000 Classes[C]// Computer Vision and Pattern Recognition. IEEE, 2014:1891-1898.

[2].Wen Y, Zhang K, Li Z, et al. A Discriminative Feature Learning Approach for Deep Face Recognition[M]// Computer Vision – ECCV 2016. Springer International Publishing, 2016:499-515.

[3].Liu W, Wen Y, Yu Z, et al. SphereFace: Deep Hypersphere Embedding for Face Recognition[J]. 2017:6738-6746.

[4].Wang F, Xiang X, Cheng J, et al. NormFace: L 2 Hypersphere Embedding for Face Verification[J]. 2017.

[5].Wang H, Wang Y, Zhou Z, et al. CosFace: Large margin cosine loss for deep face recognition[J]. arXiv preprint arXiv:1801.09414, 2018.

[6].Deng J, Guo J, Zafeiriou S. Arcface: Additive angular margin loss for deep face recognition[J]. arXiv preprint arXiv:1801.07698, 2018.

[7].F. Schroff, D. Kalenichenko and J. Philbin, FaceNet: A unified embedding for face recognition and clustering. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 815 – 823, 2015.

实践篇

本实验要求完成人脸验证，可选深度学习平台包括 Caffe,Pytorch,Tensorflow 等。这些深度学习平台集成了常用的网络层、损失函数、数据预处理、可视化等，并且提供了 Python/C++/Matlab 等多种接口。详细的介绍请参见参考书籍。

本例程的思路是：（1）用一万个人的数据库训练一个分类网络（2）用训练好的分类网络作为特征提取器，提取测试图片的特征（3）根据特征的距离进行判决，并计算出准确率。

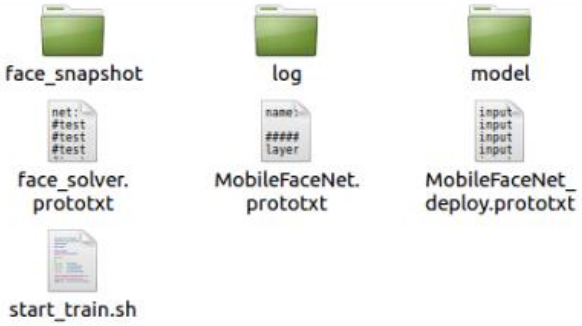
基于 Caffe 的人脸识别

Caffe 安装教程

分为 cpu 版本 caffe 和 gpu 版本 caffe。如果大家自己电脑上有显卡，建议安装 gpu 版本。
Cpu 版本 caffe 请参考 caffe 官网 http://caffe.berkeleyvision.org/
GPU 版本 caffe 请参考 caffe 官网 http://caffe.berkeleyvision.org/ 或者 https://blog.csdn.net/u014230646/article/details/51821551

训练分类网络

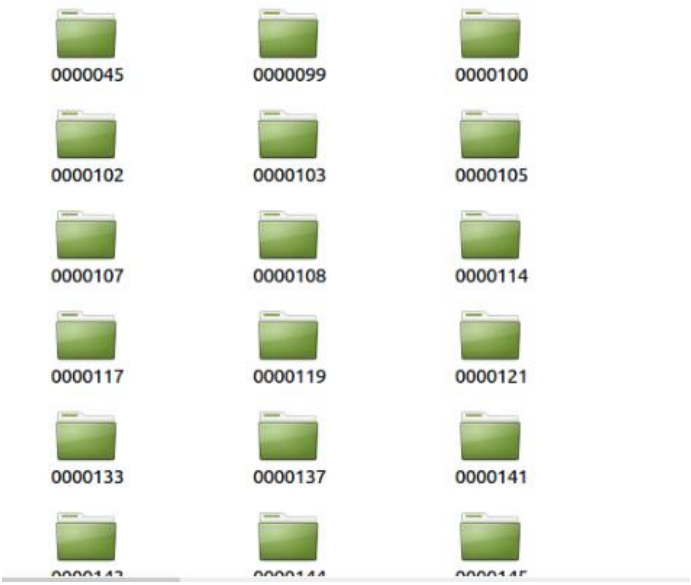
首先建立工程，我们需要网络的定义文件 mobileFaceNet.prototxt、求解器文件 face_solver.prototxt，整个工程如下：



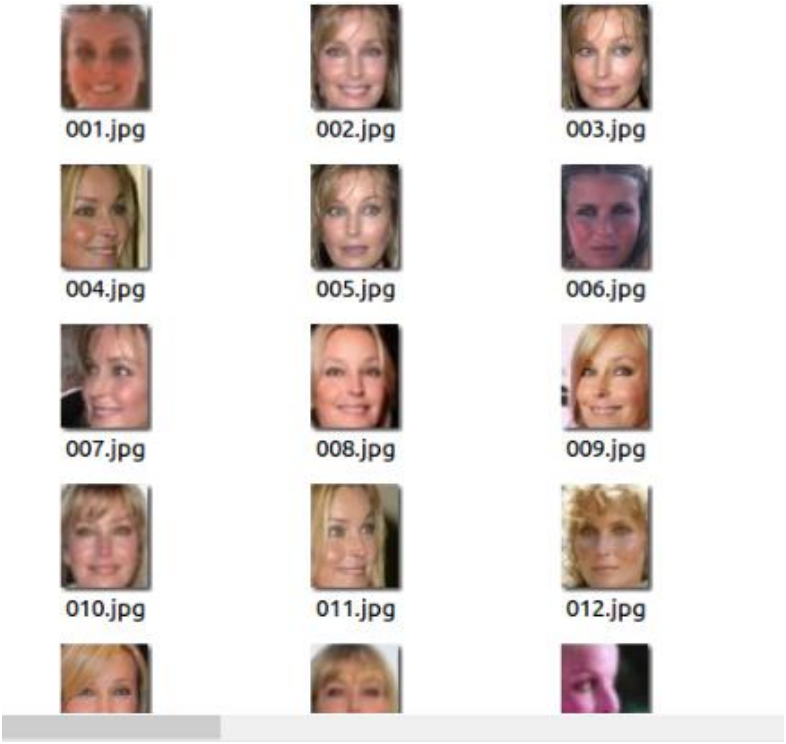
其中，model 目录保存我们训练过程中的模型参数以及训练的状态。Log 目录保存训练时候的日志输出。start_train.sh 脚本用于调用 caffe 的可执行文件，完成训练。

训练数据准备

如图，每个文件夹存放同一个人的图片：



这是某个文件中的图片：



现在我们写个 python 脚本，生成一个 train_cleaned.txt(文件名大家自己取)文件，此文件中包含了训练图片的路径。同时我们生成一个 val.txt 文件，此文件中包含了验证图片的路径，我们通过验证集来判断是否过拟合。

train_cleaned.txt 文件的格式是：每一行是一个训练样本，由训练样本的路径以及标签组成，中间用空格隔开。如下：

请注意，类别标签从 0 开始。如上图第一行表示该样本属于第 0 个类。我们提供的数据有 10575 个类别。train_cleaned.txt 文件请大家自己生成。

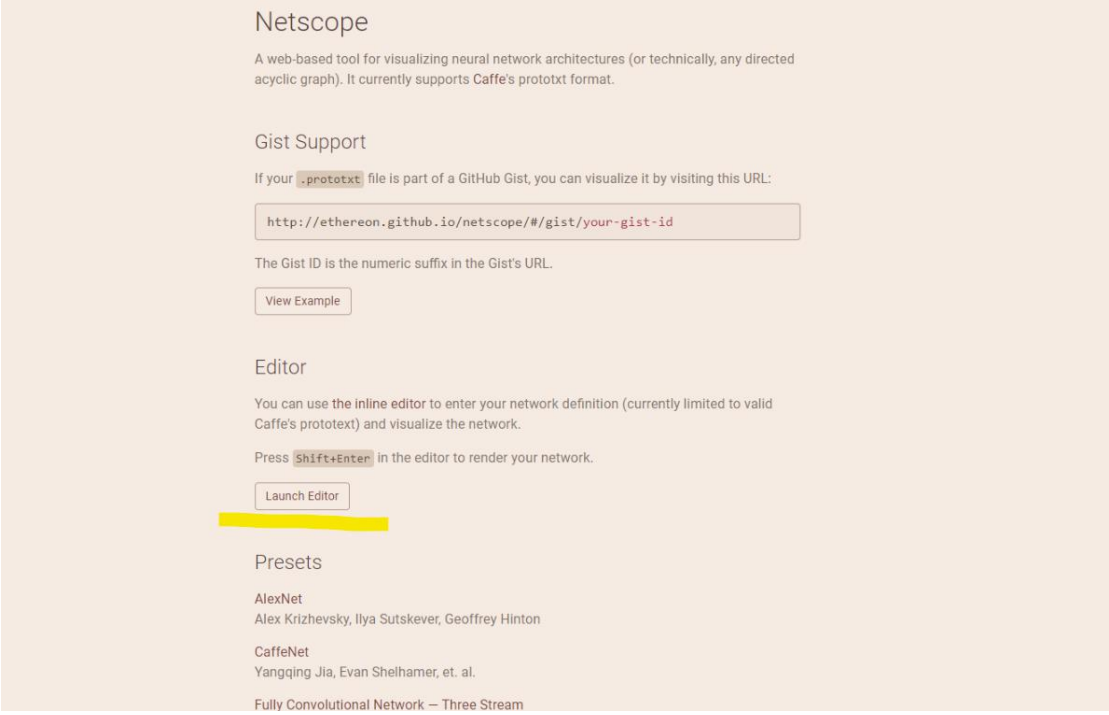
训练集和验证集的比例问题，大家可以采用：每个文件夹下随机选择 2 张图片作为验证集，其他作为训练集。

人脸识别网络结构设计

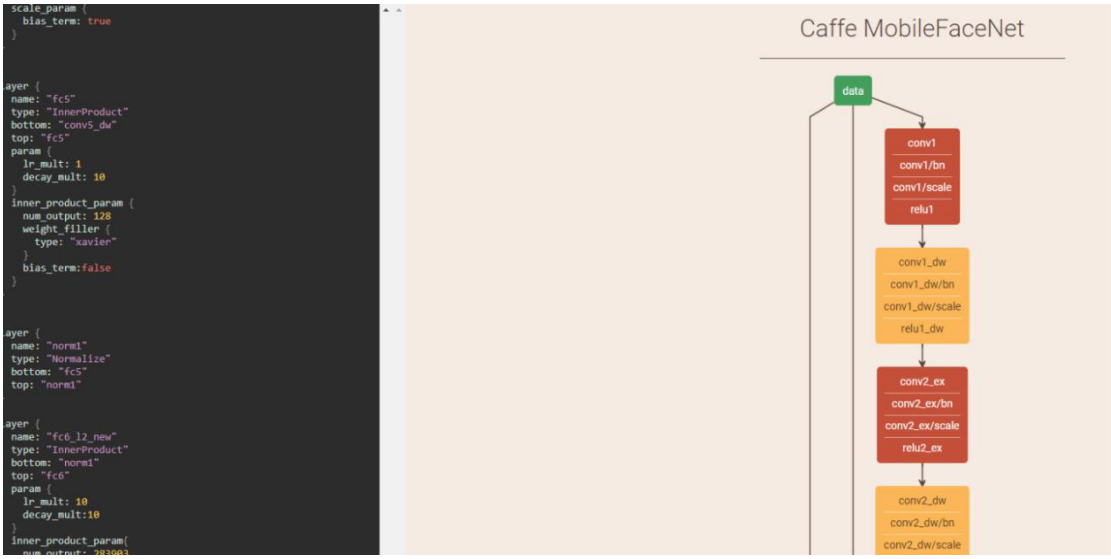
人脸识别的网络结构，大家可以借鉴一些经典的结构，如 Vgg,Resnet, DenseNet，Inception 等等。

我们在 MobileFaceNet.prototxt 中定义我们的网络结构。如下图所示是该文件的部分内容：

我们可以通过一个在线的网址对网络结构可视化 http://ethereon.github.io/netscope/quickstart.html：



输入网络并按快键键 Shift+Enter:



这个网络结构有几千行，我们可以写个 Python 脚本来生成这个网络结构，使用 Pycaffe 定义网络结构，请参考 https://blog.csdn.net/u011762313/article/details/48213421

数据输入层

在这个网络结构中，我们使用 ImageDataLayer 作为数据输入层：

```
name:"Caffe_MobileFaceNet"

############## CNN Architecture ##############
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  transform_param {
    mean_value: 127.5
    mean_value: 127.5
    mean_value: 127.5
    scale: 0.0078125
    mirror: true
  }
  image_data_param {
    root_folder: ""
    source: "/home2/chenweiliang/IDCard-Photo-0509/train_cleaned.txt"
    batch_size:64
    shuffle: true
  }
}

layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  convolution_param {
    num_output: 64
    kernel_size: 3
    stride: 2
    pad: 1
    weight_filler {
      type: "xavier"
    }
    bias_term:false
  }
```

并且指定 source，即存放训练图片路径的文本图片的路径；指定一些变换参数，这里包括减均值以及归一化，并且设置镜像图片增强（mirror: true）等。

损失函数层

这里我们使用普通的交叉熵损失函数

```
layer{
  name: "softmax_loss"
  type: "SoftmaxWithLoss"
  bottom: "fc6_margin_scale"
  bottom:"label"
  top: "softmax_loss"
}
```

在 Caffe 中，Softmax 归一化跟交叉熵损失函数是合并成一个层的。

求解器文件

主要设置学习率、动量、测试迭代次数、最大迭代次数、权重衰减、保存路径等参数。

```
net: "MobileFaceNet.prototxt"
#test_iter: 100
#test_interval: 500
#test_iter: 10
#test_interval: 500000
#test_initialization: false

#base_lr: 0.001
#base_lr: 0.00001
base_lr: 0.001
lr_policy: "fixed"
#lr_policy: "multistep"
#gamma: 0.1

#stepvalue: 80000
#stepvalue: 120000
#stepvalue: 140000
max_iter:  160000
iter_size: 1

display: 100
momentum: 0.9
weight_decay: 0.0005
snapshot: 5000
snapshot_prefix: "model/MobileFaceNet_28w_try1_2"

solver_mode: GPU
```

训练脚本

我们写一个训练脚本来开始训练。这里我们把输出的日志保存到 log 目录下。关于 Caffe 可执行文件的命令行参数的意义，请在 caffe 的根目录下查看 tools/caffe.cpp 文件。

```
#!/bin/bash

CAFFE=~/caffe-windows-ms/build/tools/caffe
TOOLS=$CAFFE/build/tools
LOG=log/step1.log



#-----------------------------------28w Cleaned Data---------------------------


# lr=0.01
#$CAFFE train --weights=./face_snapshot/MobileFaceNet.caffemodel --solver=./face_solver.prototxt --gpu 2 2>&1  | tee ./log/try1.log
# lr=0.01
#$CAFFE train --weights=./model/MobileFaceNet_28w_try1_iter_160000.caffemodel --solver=./face_solver.prototxt --gpu 2 2>&1  | tee ./log/try1_1.log
# lr=0.001
$CAFFE train --weights=./model/MobileFaceNet_28w_try1_1_iter_60000.caffemodel --solver=./face_solver.prototxt --gpu 2 2>&1  | tee ./log/try1_2.log
```

注意事项

关于网络结构定义中这些层参数的含义以及 solver 参数的含义，请在 caffe 的根目录下查看 src/caffe/proto/caffe.proto 文件。如果看了此文件还是不清楚具体的含义，请直接看对应的 layer 的源代码。

训练模型

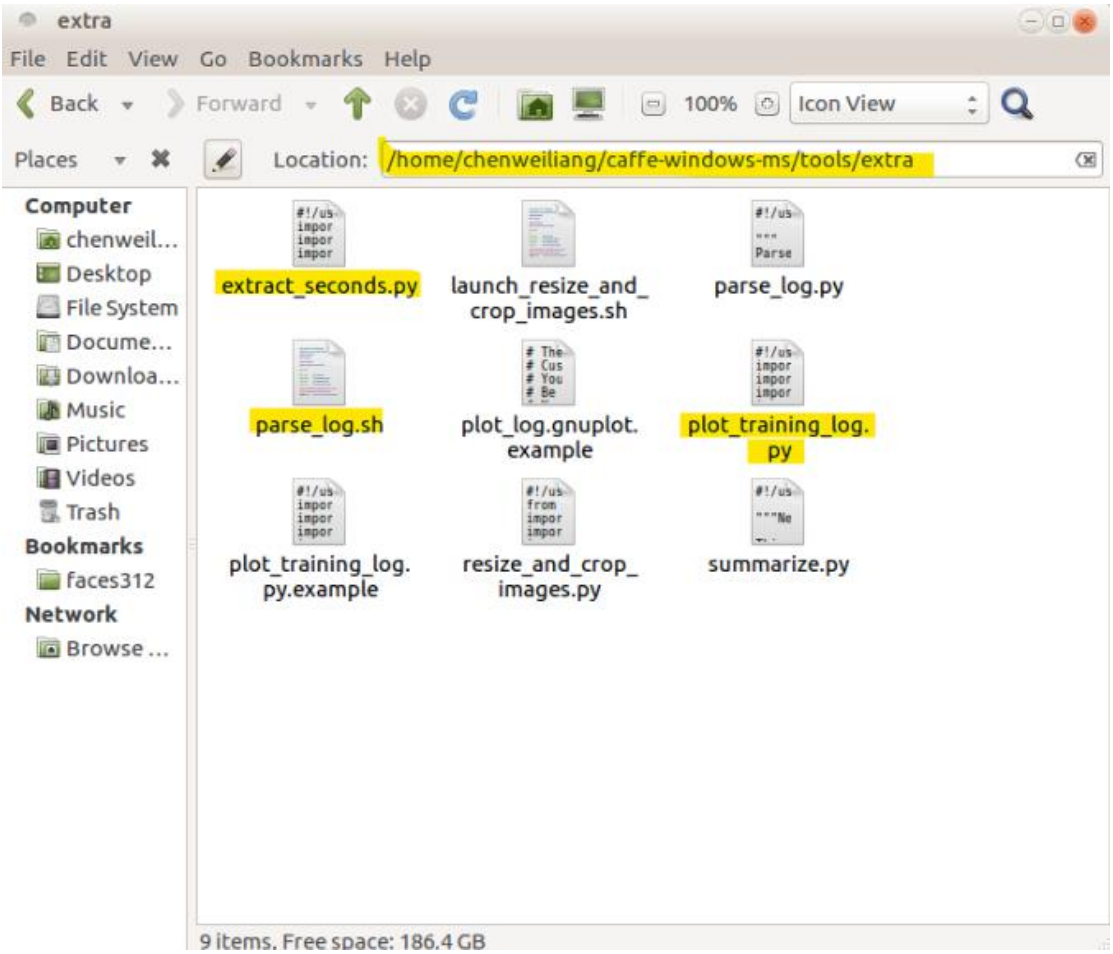
在人脸识别工程目录下运行训练脚本 start_train.sh。可以看到终端不断有 log 输出，此时这些日志输出也会保存到我们之前设置的文件中。打开 log/xxxxx.log 文件可以看到，网络初始化的过程，如图

```
I0625 09:51:53.181648 28802 caffe.cpp:223] Using GPUs 0
I0625 09:51:53.182416 28802 caffe.cpp:228] GPU 0: GeForce GTX 1080
I0625 09:51:53.363585 28802 solver.cpp:44] Initializing solver from parameters:
base_lr: 0.0001
display: 100
max_iter: 160000
lr_policy: "fixed"
momentum: 0.9
weight_decay: 0.0005
snapshot: 5000
snapshot_prefix: "model/MobileFaceNet_28w_try3_1"
solver_mode: GPU
device_id: 0
net: "MobileFaceNet.prototxt"
train_state {
  level: 0
  stage: ""
}
iter_size: 1
I0625 09:51:53.363746 28802 solver.cpp:87] Creating training net from net file: MobileFaceNet.prototxt
I0625 09:51:53.368000 28802 upgrade_proto.cpp:77] Attempting to upgrade batch norm layers using deprecated params: MobileFaceNet.prototxt
I0625 09:51:53.368018 28802 upgrade_proto.cpp:80] Successfully upgraded batch norm layers using deprecated params.
I0625 09:51:53.370285 28802 net.cpp:51] Initializing net from parameters:
name: "Caffe_MobileFaceNet"
state {
  phase: TRAIN
  level: 0
  stage: ""
}
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  transform_param {
    scale: 0.0078125
    mirror: true
    mean_value: 127.5
    mean_value: 127.5
    mean_value: 127.5
  }
  image_data_param {
    source: "/home2/chenweiliang/IDCard_Photo_0509/1_Post_112_96.txt"
```

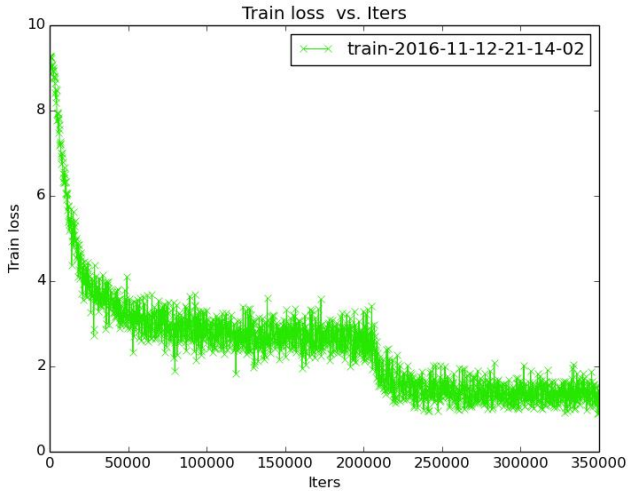
以及在训练集和测试集上的损失函数的值：

```
I0625 12:03:23.670716 28802 solver.cpp:243]     Train net output #0: softmax_loss = 5.99813 (* 1 = 5.99813 loss)
I0625 12:03:23.670753 28802 sgd_solver.cpp:137] Iteration 20200, lr = 0.0001
I0625 12:04:02.728768 28802 solver.cpp:224] Iteration 20300 (2.56034 iter/s, 39.0574s/100 iters), loss = 6.01037
I0625 12:04:02.729074 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.01037 (* 1 = 6.01037 loss)
I0625 12:04:02.729135 28802 sgd_solver.cpp:137] Iteration 20300, lr = 0.0001
I0625 12:04:41.730880 28802 solver.cpp:224] Iteration 20400 (2.56403 iter/s, 39.0011s/100 iters), loss = 5.73539
I0625 12:04:41.731210 28802 solver.cpp:243]     Train net output #0: softmax_loss = 5.73539 (* 1 = 5.73539 loss)
I0625 12:04:41.731242 28802 sgd_solver.cpp:137] Iteration 20400, lr = 0.0001
I0625 12:05:20.755738 28802 solver.cpp:224] Iteration 20500 (2.56254 iter/s, 39.0238s/100 iters), loss = 6.88714
I0625 12:05:20.756021 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.88714 (* 1 = 6.88714 loss)
I0625 12:05:20.756052 28802 sgd_solver.cpp:137] Iteration 20500, lr = 0.0001
I0625 12:05:59.848601 28802 solver.cpp:224] Iteration 20600 (2.55808 iter/s, 39.0919s/100 iters), loss = 5.76623
I0625 12:05:59.848850 28802 solver.cpp:243]     Train net output #0: softmax_loss = 5.76623 (* 1 = 5.76623 loss)
I0625 12:05:59.848877 28802 sgd_solver.cpp:137] Iteration 20600, lr = 0.0001
I0625 12:06:38.832100 28802 solver.cpp:224] Iteration 20700 (2.56525 iter/s, 38.9825s/100 iters), loss = 6.49615
I0625 12:06:38.832278 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.49615 (* 1 = 6.49615 loss)
I0625 12:06:38.832291 28802 sgd_solver.cpp:137] Iteration 20700, lr = 0.0001
I0625 12:07:17.789499 28802 solver.cpp:224] Iteration 20800 (2.56697 iter/s, 38.9565s/100 iters), loss = 6.6099
I0625 12:07:17.789809 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.6099 (* 1 = 6.6099 loss)
I0625 12:07:17.789839 28802 sgd_solver.cpp:137] Iteration 20800, lr = 0.0001
I0625 12:07:56.801331 28802 solver.cpp:224] Iteration 20900 (2.56339 iter/s, 39.0108s/100 iters), loss = 6.21324
I0625 12:07:56.801651 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.21324 (* 1 = 6.21324 loss)
I0625 12:07:56.801682 28802 sgd_solver.cpp:137] Iteration 20900, lr = 0.0001
I0625 12:08:35.737262 28802 solver.cpp:224] Iteration 21000 (2.56839 iter/s, 38.9349s/100 iters), loss = 6.49962
I0625 12:08:35.737541 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.49962 (* 1 = 6.49962 loss)
I0625 12:08:35.737578 28802 sgd_solver.cpp:137] Iteration 21000, lr = 0.0001
I0625 12:09:14.665259 28802 solver.cpp:224] Iteration 21100 (2.56891 iter/s, 38.927s/100 iters), loss = 6.63891
I0625 12:09:14.665804 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.63891 (* 1 = 6.63891 loss)
I0625 12:09:14.665827 28802 sgd_solver.cpp:137] Iteration 21100, lr = 0.0001
I0625 12:09:53.636152 28802 solver.cpp:224] Iteration 21200 (2.5661 iter/s, 38.9696s/100 iters), loss = 7.35007
I0625 12:09:53.636493 28802 solver.cpp:243]     Train net output #0: softmax_loss = 7.35007 (* 1 = 7.35007 loss)
I0625 12:09:53.636531 28802 sgd_solver.cpp:137] Iteration 21200, lr = 0.0001
I0625 12:10:32.597543 28802 solver.cpp:224] Iteration 21300 (2.56671 iter/s, 38.9604s/100 iters), loss = 6.51437
I0625 12:10:32.597715 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.51437 (* 1 = 6.51437 loss)
I0625 12:10:32.597729 28802 sgd_solver.cpp:137] Iteration 21300, lr = 0.0001
I0625 12:11:11.616904 28802 solver.cpp:224] Iteration 21400 (2.56289 iter/s, 39.0185s/100 iters), loss = 6.03859
I0625 12:11:11.617228 28802 solver.cpp:243]     Train net output #0: softmax_loss = 6.03859 (* 1 = 6.03859 loss)
I0625 12:11:11.617261 28802 sgd_solver.cpp:137] Iteration 21400, lr = 0.0001
I0625 12:11:50.645262 28802 solver.cpp:224] Iteration 21500 (2.56231 iter/s, 39.0273s/100 iters), loss = 7.54361
I0625 12:11:50.645491 28802 solver.cpp:243]     Train net output #0: softmax_loss = 7.54361 (* 1 = 7.54361 loss)
I0625 12:11:50.645504 28802 sgd_solver.cpp:137] Iteration 21500, lr = 0.0001
I0625 12:12:29.677984 28802 solver.cpp:224] Iteration 21600 (2.56202 iter/s, 39.0318s/100 iters), loss = 6.51152
```

现在，我们来写一个脚本，把这些 log 解析出来，然后画出损失函数的曲线，linux 系统有许多方便的命令可以快速根据关键字查找匹配的字段。Caffe 已经提供了这样的工具，在 Caffe 根目录下，我们可以看到有 tools/extra 文件夹，包含了一些有用的工具，如解析 log 文件的 parse_log.sh 脚本，以及画出损失函数曲线的 plot_training_log.py.example 脚本。



使用 plot_training_log.py.example 脚本可以画出损失函数曲线，请查看 plot_training_log.py.example 的源码。最终如图：

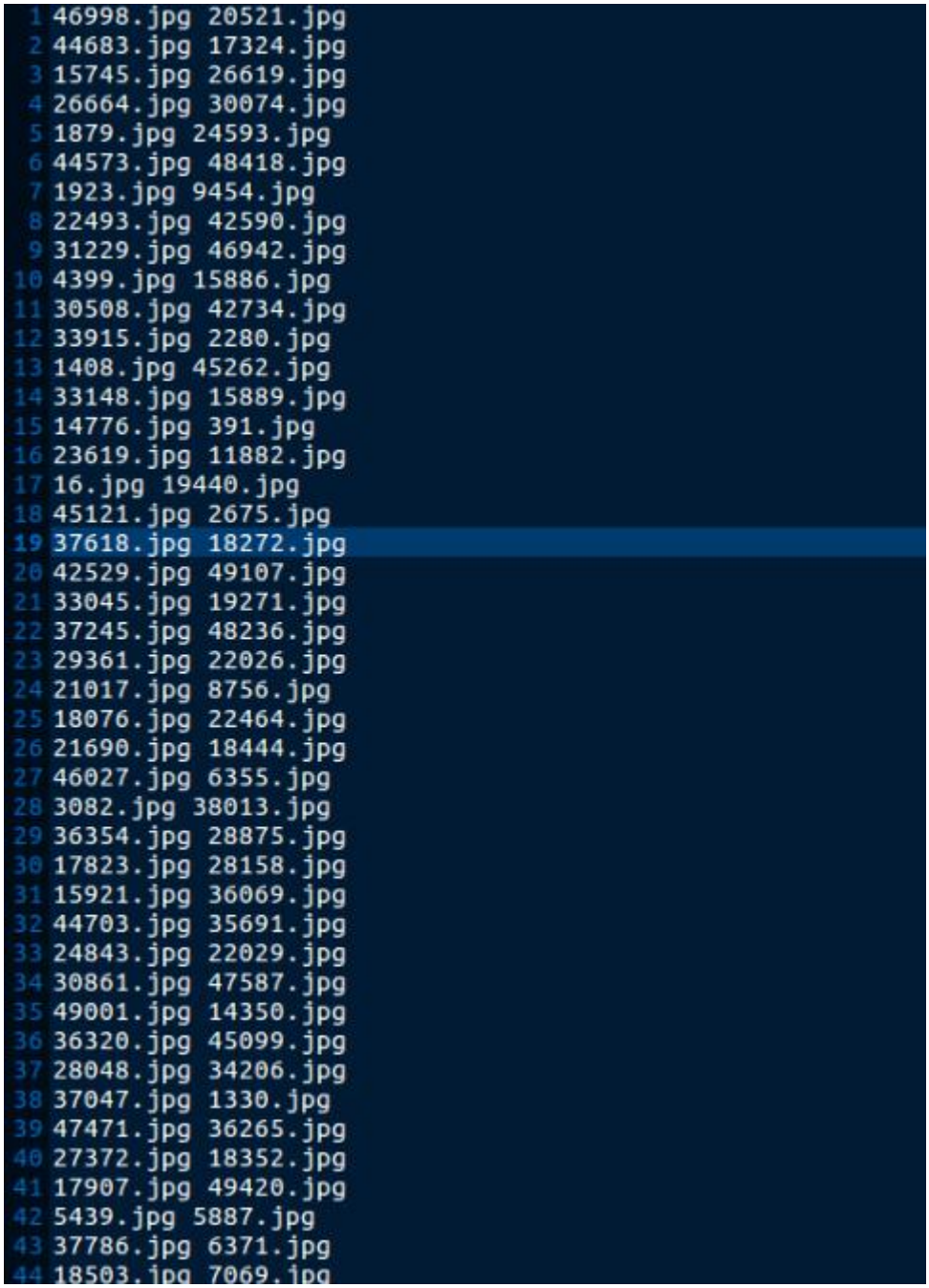


请注意，由于网络结构、batch_size、损失函数设置的不相同，你训练出来的损失函数曲线可能并不与上图相同。

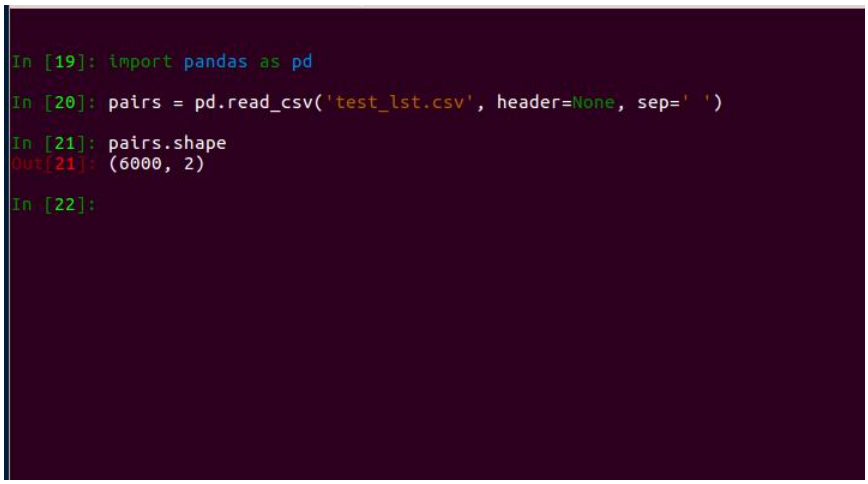
测试

训练完之后，我们针对测试数据，用我们训练好的网络对每一对图片提取特征，根据特征的

距离去判断这一对图片是否属于同一个人。因为我们的网络是采用人脸分类作为监督训练的，所以我们认为在特征空间中同一个人的图片距离近，不同人的距离远。因此根据特征距离的远近去判断两张测试图片是否属于同一个人。
如图是测试的图片列表 test_lst.csv，每一行两张图片，判断是否为同一个人：



使用 python 读取测试图片列表 test_lst.csv:



然后对每张图片提取特征，得到其特征向量。再根据特征向量的距离去判断每对图片是否是属于同一个人，并且把预测的结果写到 result.csv 文件里。

关于如何用 pycaffe 提取图片特征，请参考 https://www.cnblogs.com/louyihang-loves-baiyan/p/5078746.html

基于 Pytorch 的人脸识别

请到网络上搜索资料。

基于 TensorFlow 的人脸识别

请到网络上搜索资料。

参考资料

[1] https://blog.csdn.net/a_1937/article/details/50334919
[2] https://github.com/goodluckcwl/Face-Verification
[3] https://github.com/happynear/FaceVerification