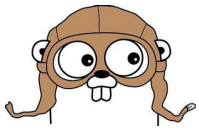


验证码破解技术四部曲之使用卷积神经网络（四）

Sep 23, 2016

前言

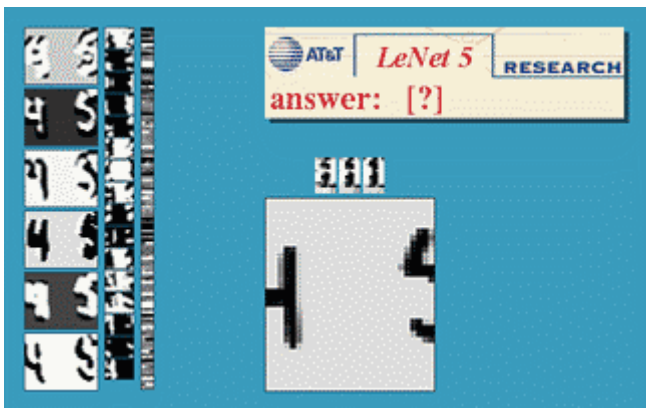
在这节，我将用卷积神经网络（简称：CNN）破解新浪微博手机端的验证码(http://login.weibo.cn/login/)，验证码如下。

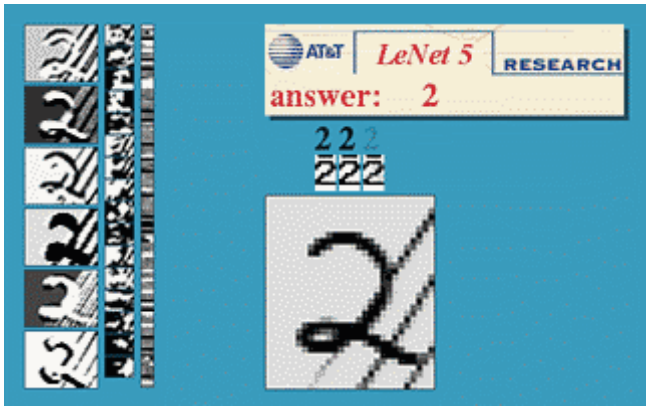
本节的代码可以在https://github.com/nladuo/captcha-break/tree/master/weibo.cn找到。

关于神经网络的原理很难在一节讲清楚。在这里，只需要把神经网络当成一个黑匣子，输入是一个图片，输出一个label，也就是类别。

LeNet5

本节使用的神经网络是国外学者Yann LeCun的LeNet5，该神经网络以32x32的图片作为输入，对于字符的变形、旋转、干扰线等扭曲都可以很好的识别，可以实现以下效果。





更多的效果可以在http://yann.lecun.com/exdb/lenet/上查看，具体原理可以查看Yann LeCun的论文。

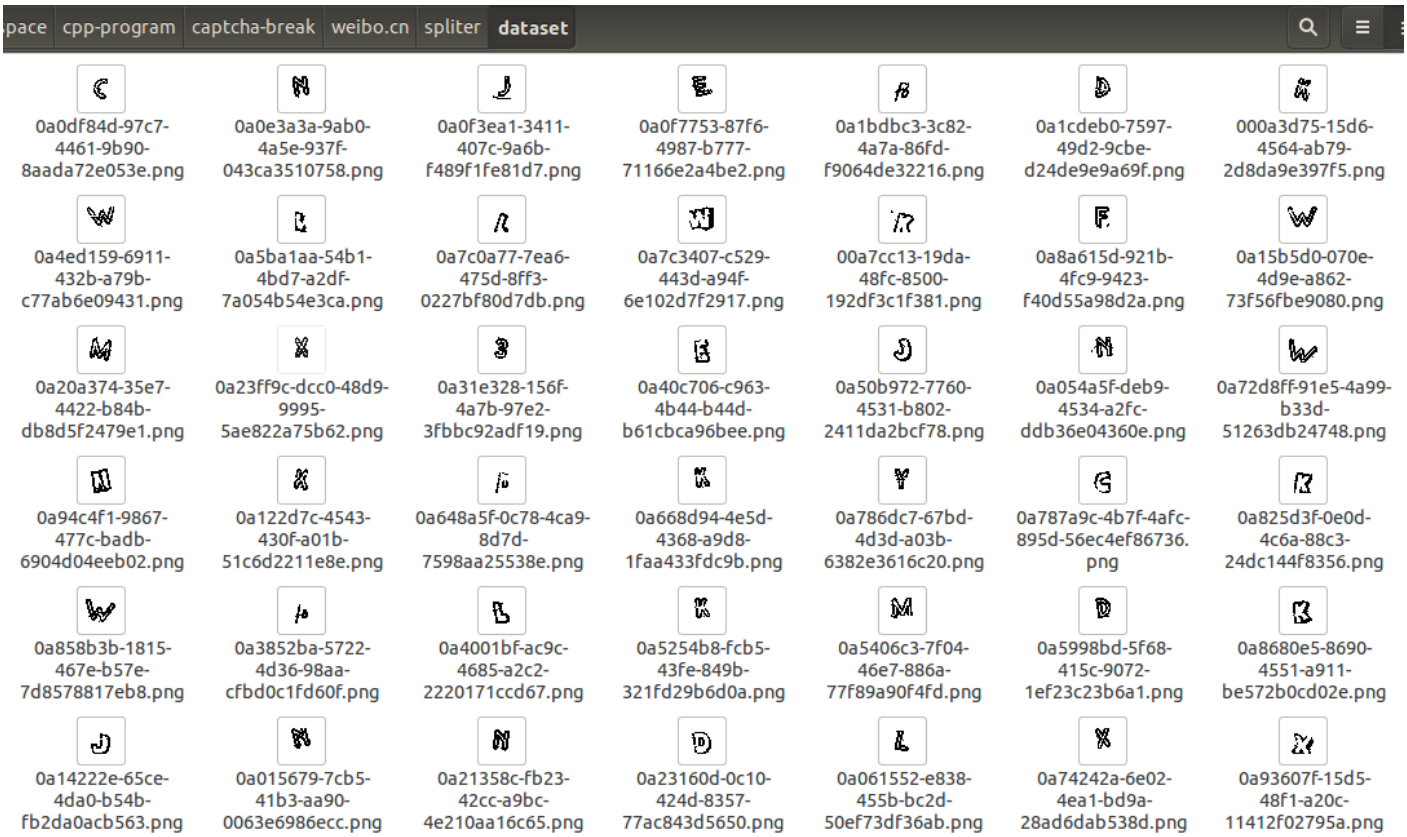
字符下载

字符下载和上节差不多，这里需要注意的是新浪微博的验证码下载下来是gif格式的，opencv不支持读取gif的读取，需要用PIL把验证码转换成png格式。

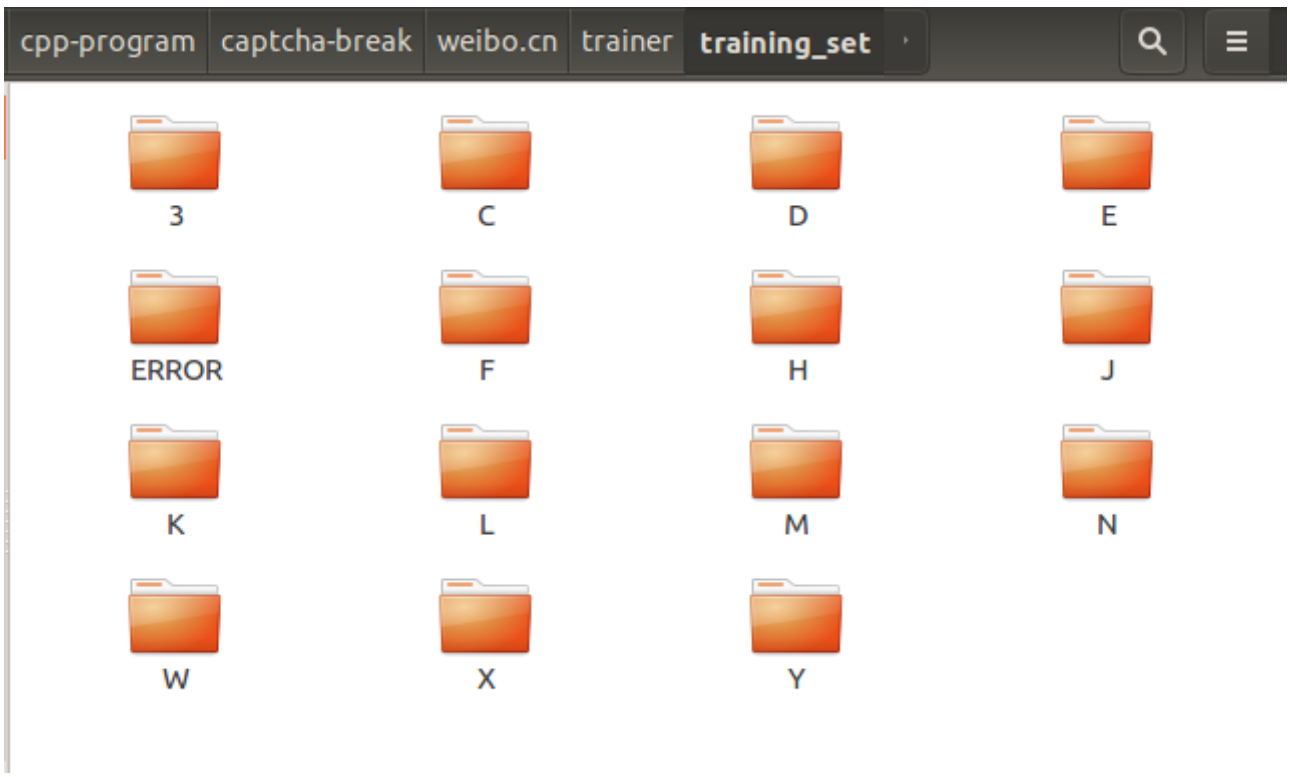
另外，新浪微博的验证码明显比CSDN下载的验证码要复杂得多，所以需要大量的样本，至少要下载上千个验证码。

字符分割

新浪微博的验证码需要进行去除椒盐噪声、去除干扰线、二值化后，才能很好的进行垂直投影分割，我算法写的不是很好，就不在这里展开了，代码可以在spliter中找到。LeNet5的输入是32x32像素，所以为了不对神经网络进行大量修改，也需要将每个字母都方法32*32的模板中，分割后如下：



分割好之后，需要开始大量的人工操作了，经过了几个小时的努力，成功完成了5000多样本的分类，结果放在了trainer/training_set中。



这里每个文件夹都是一个分类，共有14个分类（除了ERROR），点进文件夹后可以看到每个文件夹内都有300多张图片。

训练

构建网络

我这里使用的神经网络库是tiny-cnn（现在已改名叫tiny-dnn）。

训练相关的代码都在trainer/main.cpp中，首先看一下神经网络的构造函数。

C++

```
void construct_net(network<sequential>& nn) {
    // connection table [Y.Lecun, 1998 Table.1]
#define O true
#define X false
    static const bool tbl[] = {
        O, X, X, X, O, O, O, X, X, O, O, O, O, X, O, O,
        O, O, X, X, X, O, O, O, X, X, O, O, O, O, X, O,
        O, O, O, X, X, X, O, O, O, X, X, O, X, O, O, O,
        X, O, O, O, X, X, O, O, O, O, X, X, O, X, O, O,
        X, X, O, O, O, X, X, O, O, O, O, X, O, O, X, O,
        X, X, X, O, O, O, X, X, O, O, O, O, X, O, O, O
    };
```

```
#undef O
#undef X

    // construct nets
    nn << convolutional_layer<tan_h>(32, 32, 5, 1, 6)  // C1, 1@32x32-in, 6@28x28-out
       << average_pooling_layer<tan_h>(28, 28, 6, 2)   // S2, 6@28x28-in, 6@14x14-out
       << convolutional_layer<tan_h>(14, 14, 5, 6, 16,
            connection_table(tbl, 6, 16))              // C3, 6@14x14-in, 16@10x10-in
       << average_pooling_layer<tan_h>(10, 10, 16, 2)  // S4, 16@10x10-in, 16@5x5-out
       << convolutional_layer<tan_h>(5, 5, 5, 16, 120) // C5, 16@5x5-in, 120@1x1-out
       << fully_connected_layer<tan_h>(120, 14);       // F6, 120-in, 14-out
}
```

这里可以看到有六层神经网络，C1、S2、C3、S4、C5、F6。其实不用仔细的了解神经网络的构造，只需要把它想象成一个黑匣子，黑匣子的输入就是C1层的输入（C1, 1@32x32-in），黑匣子的输出就是F6层（F6,14-out）。32x32对应着图片的大小，14对应着类的个数。比如说要训练MINST数据集（一个手写字符的数据集）的话，需要把fully_connected_layer(120, 14)改成fully_connected_layer(120, 10)，因为MINST中有十类字符(0-9十种数字)。

（注：这里只能修改F6层的参数而不能修改C1层的参数，修改C1参数会影响到其他层的输入。）

加载数据集

接下来，通过boost库加载数据集，其中五分之四的样本作为训练，还有五分之一的作为测试训练的正确性。

C++

```
std::string label_strs[14] = {
    "3", "C", "D", "E", "F", "H", "J", "K", "L", "M", "N", "W", "X", "Y"
};
void load_dataset(std::vector<label_t> &train_labels,
                  std::vector<vec_t> &train_images,
                  std::vector<label_t> &test_labels,
                  std::vector<vec_t> &test_images)
{
    for (int i = 0; i < 14; ++i){
        std::vector<std::string> images;

        fs::directory_iterator end_iter;
        fs::path path("./training_set/"+label_strs[i]);
        for (fs::directory_iterator iter(path); iter != end_iter; ++iter){
            if (fs::extension(*iter)==".png"){
                images.push_back(iter->path().string());
            }
        }

        //train_set.size() : test_set.size() = 4:1
        int flag = 0;
        std::vector<std::string>::iterator itr = images.begin();
        for (;itr != images.end(); ++itr){
            vec_t data;
            convert_image(*itr, -1.0, 1.0, 32, 32, data);
```

```
            if (flag <= 4){
                train_labels.push_back(i);
                train_images.push_back(data);
            }else{
                test_labels.push_back(i);
                test_images.push_back(data);
                flag = 0;
            }
            flag++;
        }
    }
}
```

参数设置

卷积神经网络使用的是随机梯度下降进行训练，涉及一些数学知识，这里就不展开了。

这里只要把它理解为：神经网络会自己不断的对数据集进行学习(不断的迭代，每次迭代都会对识别率有所改进)。学习的过程会有一个学习速率optimizer.alpha，这里选择的是默认的；还有每次学习多少个数据(minibatch_size)，这里设置每次对100个数据进行学习；还有一个学习的时间(num_epochs)，这里学习了50次之后，学习效果就没有了。也就是识别率达到了峰值。

C++

```
int minibatch_size = 100;               //每批量的数量
int num_epochs = 50;                    //迭代次数

// optimizer.alpha *= std::sqrt(minibatch_size); 使用默认的学习速率
```

保存结果

神经网络的训练之后，需要保存神经网络的权重，把权重输出到”weibo.cn-nn-weights”中。

C++

```
// save networks
std::ofstream ofs("weibo.cn-nn-weights");
ofs << nn;
```

运行程序

运行trainer后，可以看到开始加载数据，并且进行一次一次的迭代，每一次迭代都会根据测试数据来进行验证，显示正确识别的字符数目。

从上面可以看到，一共有3934个训练样本和972个测试样本，正确识别的字符数目随着迭代次数不断的增加，从72->120->142->223....，识别率不断增加。

训练到最后(第四十几次迭代)，可以看到数据已经差不多饱和了,维持在860、870左右，也就是单个字符有89%的识别率，单个验证码有0.89^4=0.64左右的识别率。(如果训练了很多次后，发现识别率还没有饱和，可以增大迭代次数num_epochs或者增大学习速率optimizer.alpha)

识别

最后，可以通过训练好的“weibo.cn-nn-weights”来进行识别，把trainer/weibo.cn-nn-weights放到recognizer文件夹下。

接下来看看神经网络是如何进行识别的，在recognizer/main.cpp中查看recognize函数。

```
int recognize(const std::string& dictionary, cv::Mat &img) {
    network<sequential> nn;

    construct_net(nn);

    // load nets
    ifstream ifs(dictionary.c_str());
    ifs >> nn;

    // convert cvMat to vec_t
    vec_t data;
    convert_mat(img, -1.0, 1.0, 32, 32, data);

    // recognize
    auto res = nn.predict(data);
    vector<pair<double, int> > scores;
```

```
    for (int i = 0; i < 14; i++)
        scores.emplace_back(rescale<tan_h>(res[i]), i);

    // sort and get the result
    sort(scores.begin(), scores.end(), greater<pair<double, int>>());
    return scores[0].second;
}
```

在神经网络的最后一层中输出的是一个14维的向量，分别对应着每个类的概率，所以通过sort函数，找出概率最大的类就是识别结果了。

测试图片：


test1.png


test2.png


test3.png


test4.png

测试识别结果：

```
kalen@kalen-ThinkPad-S3-S431:~/Workspace/cpp-program/captcha-break/weibo.cn/recognizer$ ./recognizer test1.png
Result:HEJY
kalen@kalen-ThinkPad-S3-S431:~/Workspace/cpp-program/captcha-break/weibo.cn/recognizer$ ./recognizer test2.png
Result:WLJK
kalen@kalen-ThinkPad-S3-S431:~/Workspace/cpp-program/captcha-break/weibo.cn/recognizer$ ./recognizer test3.png
Result:XEEF
kalen@kalen-ThinkPad-S3-S431:~/Workspace/cpp-program/captcha-break/weibo.cn/recognizer$ ./recognizer test4.png
Result:3HDE
```

NEXT

0条评论

最新 最早 最热

还没有评论，沙发等你来抢

True's ME正在使用多说