Computer Vision Exp8

刘钧元 15310022

测试环境:

整个程序于 mac OS 10.13.3 下开发，所调用的外部库包括 CImg 与 X11。

测试数据:

程序的功能包括:

1. 根据标准流程输出每一个主要步骤的结果，包括 A4 纸张的矫正结果，行数据 (包括学号、手机号、身份证号)的切割，单个字符的切割结果

2. 对上面的 A4 纸的四个角、学号、手机号、身份证号进行识别, 识别结果保存到 Excel 表格 (xlsx 格式), 对于手写体数字字符的训练数据可以使用 MNIST。

测试结果及分析:

1. 采用边缘检测或者图像分割的方法获取图像的边缘，并计算图像的四个角点，完成图像矫正。

在此，我们使用上一次实验的代码来完成图像矫正。ImageCorrection 类的接口与上次实验中类似，唯一不同点在于增加了 binary() 函数来完成二值化。由于 correct() 函数并没有做太多修改，这部分的内容就不再赘述了。详细的矫正结果可见 Correction 文件夹。

```
#ifndef _IMAGE_CORRECTION_HPP_
#define _IMAGE_CORRECTION_HPP_

#include <iostream>
#include <vector>
#include <cmath>
#include <stack>
#include <algorithm>
#include <fstream>
#include <CImg.h>
#include <Matrix.hpp>

#define uchar unsigned char


class ImageCorrection {
public:
    static cimg_library::CImg<uchar> rgb2grey(const cimg_library::CImg<uchar>&);
    static uchar otsu(const cimg_library::CImg<uchar>&);
    static cimg_library::CImg<short> hough(const cimg_library::CImg<uchar>&, const uchar);
    static std::vector<std::tuple<int, int, int>> vote(const cimg_library::CImg<short>&, const int, const int, const int = 50, const int = 50);
    static cimg_library::CImg<uchar> correct(const cimg_library::CImg<uchar>&, const int sn);
    static cimg_library::CImg<uchar> binary(const cimg_library::CImg<uchar>&, const int = 139, const double = 0.90);

private:
    static cimg_library::CImg<uchar> draw(const cimg_library::CImg<uchar>&, const uchar);
    static cimg_library::CImg<uchar> draw(const cimg_library::CImg<uchar>&, const std::vector<std::tuple<int, int, int>>&);
    static cimg_library::CImg<uchar> warp(const cimg_library::CImg<uchar>&, const std::vector<std::tuple<int, int, int>>&);
    static cimg_library::CImg<uchar> crop(const cimg_library::CImg<uchar>&);
};


#endif
```

与上次实验相比，我在这次实验中往 vote() 函数中加入了筛选模型的代码。我们首先筛选掉 sin(alpha) == 0.0 的模型，防止我们在之后得到斜率与截距皆不存在的直线。之后，我们再按照模型的 frequency 进行排序，只留下投票数最高的 4 个模型作为 A4 纸的边缘。通过这样的操作，我们对所有的输入都能获得较好的输出结果。

```
// eliminate rebundant model(s)
if (models.size() > 4) {
    #define point tuple<int, int, int>
    for (vector<point>::iterator it = models.begin(); it != models.end(); ) {
        double alpha = (double(get<0>(*it))/180) * M_PI;
        if (sin(alpha) == 0.0) {
            it = models.erase(it);
        } else {
            ++it;
        }
    }
    sort(models.begin(), models.end(), [&](const point& a, const point& b) -> bool { return get<2>(a) > get<2>(b); } );
    while (models.size() > 4) {
        models.pop_back();
    }
}
```

2. 采用图像分割（二值化）的方法，获取图像中的手写字符，输出二值化结果。

我们通过调用 ImageCorrection 中的 binary() 函数来完成二值化。对于 binary()，我们需要读入矫正后的 A4 纸作为输入。我们把上一步的结果存放在 Correction 文件夹中，然后 binary() 读入 Correction 中的矫正后的 A4 纸，完成二值化后，将结果保存到 Binary 文件夹中。

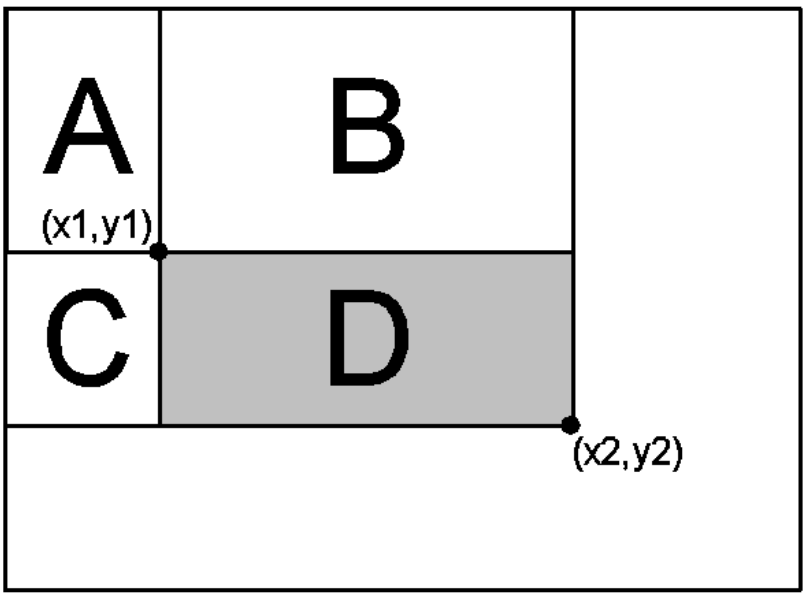
```
CImg<uchar> ImageCorrection::binary(const CImg<uchar>& src, const int block, const double ratio) {
    CImg<long> p(src);
    for (int i = 0; i < p.height(); ++i) {
        for (int j = 1; j < p.width(); ++j) {
            p(j, i) += p(j-1, i);
            if (i != 0) {
                p(j-1, i) += p(j-1, i-1);
            }
        }
        if (i != 0) {
            p(p.width()-1, i) += p(p.width()-1, i-1);
        }
    }
    CImg<uchar> dst(src);
    cimg_forXY(dst, x, y) {
        int x1 = max(0, x - block/2), x2 = min(dst.width()-1, x + block/2);
        int y1 = max(0, y - block/2), y2 = min(dst.height()-1, y + block/2);
        int count = (x2 - x1 + 1) * (y2 - y1 + 1);
        double sum = p(x2, y2);
        if (x1 != 0 && y1 != 0) {
            sum -= p(x2, y1-1) + p(x1-1, y2) - p(x1-1, y1-1);
        } else if (x1 != 0) {
            sum -= p(x1-1, y2);
        } else if (y1 != 0) {
            sum -= p(x2, y1-1);
        }
        if (dst(x, y) < ratio*sum/count) {
            dst(x, y) = 0;
        } else {
            dst(x, y) = 255;
        }
    }

    return ImageCorrection::crop(dst);
}
```

binary() 的实现如图。这里，我们没有继续使用 Otsu 算法，我们使用的是局部自适应阀值来完成图像分割。因为输入的图片可能会有闪光灯或者阴影的之类的干扰存在，使用 Otsu 算法的话不能很好地提取出图片中的文字。我们使用了一种局部自适应快速二值化方法，首先我们要计算出图像每个点的左上方所有点的像素值之和。

4	1	2	2
0	4	1	3
3	1	0	4
2	1	3	2

4	5	7	9
4	9	12	17
7	13	16	25
9	16	22	33

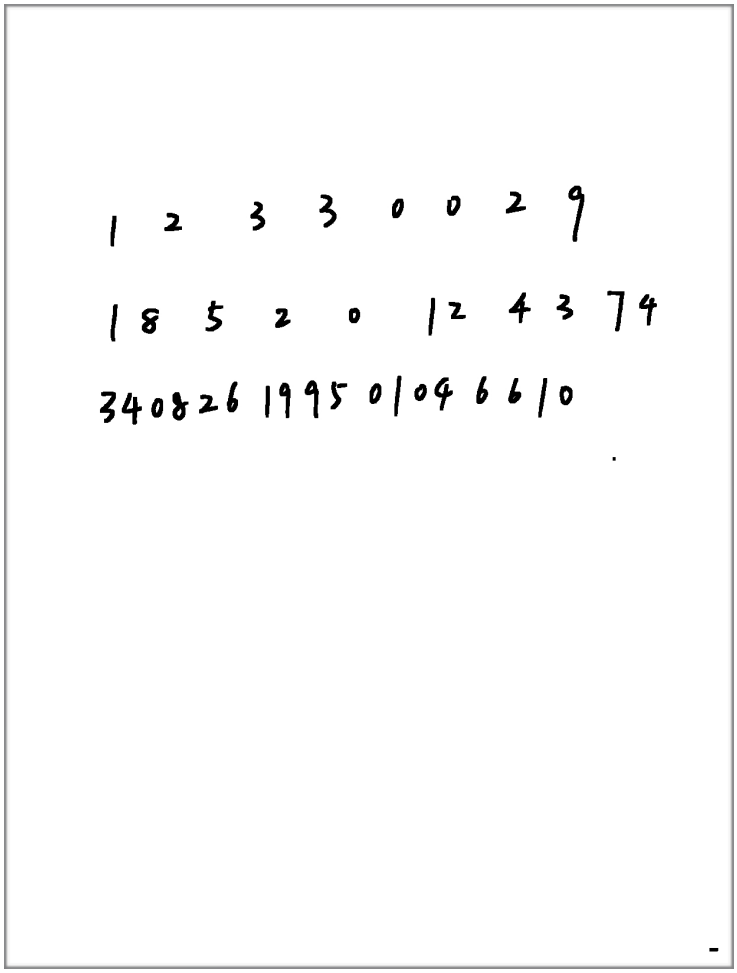
上图为计算出图像每个点的左上方所有点的像素值之和的一个例子，左图为输入图片，右图为我们需要的结果。



在进行了上述预操作后，我们就可以快速地计算出每一个块的像素值和。对于上图中的区域 D 有，P(D) = P(x2, y2) - P(x2, y1) -P(x1, y2) + P(x1, y1)。binary() 函数需要传入两个参数，其中：block 为块的大小，ratio 为二值化的阈值比率。对于图像中的每一个块，我们都进行如下判定：如果图像中心点的像素值 < 块的平均像素值乘以二值化阈值比率，我们就将中心点像素设为 0，否则设为 255。这样，我们就能获得文字为黑色，背景为白色的二值化图像。

```
CImg<uchar> ImageCorrection::crop(const cimg_library::CImg<uchar>& src) {
    CImg<uchar> dst(src);
    stack<pair<int, int>> s;
    int width = dst.width(), height = dst.height();
    for (int i = 0; i < width; ++i) {
        if (dst(i, 0) == 0) {
            s.push(make_pair(i, 0));
        }
        if (dst(i, height-1) == 0) {
            s.push(make_pair(i, height-1));
        }
    }
    for (int i = 0; i < height; ++i) {
        if (dst(0, i) == 0) {
            s.push(make_pair(0, i));
        }
        if (dst(width-1, i) == 0) {
            s.push(make_pair(width-1, i));
        }
    }
    while (!s.empty()) {
        pair<int, int> p = s.top();
        s.pop();
        for (int i = 0; i < 8; ++i) {
            int a = p.first + dirs[i][0], b = p.second + dirs[i][1];
            if (a >= 0 && a < width && b >= 0 && b < height) {
                if (dst(a, b) == 0) {
                    s.push(make_pair(a, b));
                }
            }
        }
        dst(p.first, p.second) = 255;
    }

    return dst;
}
```

但到这一步为止，我们仍未能获得理想的二值化图片。因为图像矫正的结果可能会留有一部分的边缘未能截去，所以二值化之后，这部分的干扰将依然存在。因此，我们使用深搜的方法，把图像边缘的连通黑色像素点设为白色，防止这些像素干扰字符切割的结果。

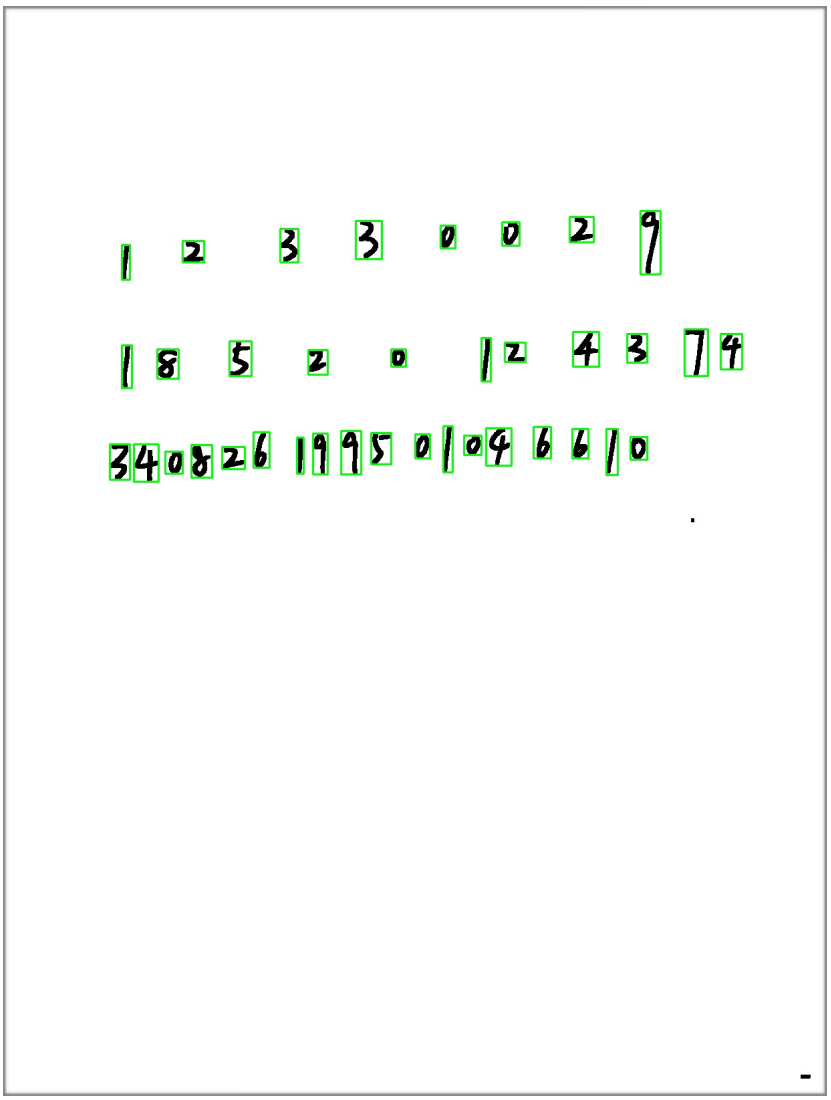


为了获得更好的结果，我们在二值化之前进行了腐蚀操作，使图像中的文字变粗，其中的一个结果如图，详细的结果保存在 Binary 文件夹中。

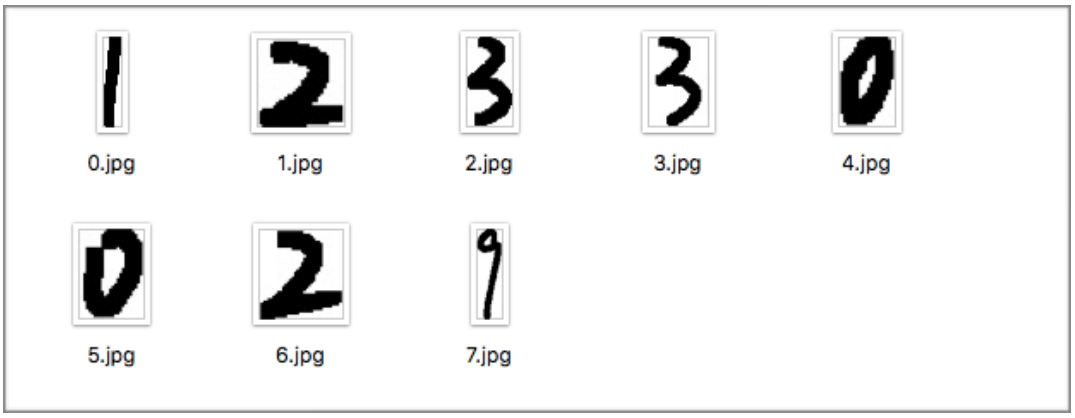
3. 针对二值化图像，对 Y 方向投影切割出各个行的图像，例如学号切成单个的图像，手机号和身份证号也如此。针对行图像（如学号图像），对 X 方向做投影切割，切割出单个字符。

```python
def getContours():
    for i in range(0, 63):
        """ load an image """
        img = cv.imread('Binary/%d.jpg' % i)
        """ convert the source to greyscale """
        grey = 255 - cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        # kernel = np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1]], np.uint8)
        # grey = cv.erode(cv.dilate(grey, kernel, iterations=3), kernel, iterations=2)
        """ find contours """
        binary, contours, hierarchy = cv.findContours(grey, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
        """ crop digit(s) """
        if not os.path.isdir('Segmentation/%d' % i):
            os.mkdir('Segmentation/%d' % i)
        contours = [contour for contour in contours if cv.contourArea(contour) >= 96]
        contours.sort(key=lambda contour: cv.minEnclosingCircle(contour)[0][1])
        """ number of group(s) """
        group = 0
        while group*37 < contours.__len__():
            """ student id """
            if not os.path.isdir('Segmentation/%d/%d' % (i, group*3)):
                os.mkdir('Segmentation/%d/%d' % (i, group*3))
            n = 0
            for contour in sorted(contours[group*37:group*37+8], key=lambda contour: cv.minEnclosingCircle(contour)[0][0]):
                [x, y, w, h] = cv.boundingRect(contour)
                cv.imwrite('Segmentation/%d/%d/%d.jpg' % (i, group*3, n), img[y:y+h, x:x+w])
                n += 1
            """ tel number """
            if not os.path.isdir('Segmentation/%d/%d' % (i, group*3+1)):
                os.mkdir('Segmentation/%d/%d' % (i, group*3+1))
            n = 0
            for contour in sorted(contours[group*37+8:group*37+19], key=lambda contour: cv.minEnclosingCircle(contour)[0][0]):
                [x, y, w, h] = cv.boundingRect(contour)
                cv.imwrite('Segmentation/%d/%d/%d.jpg' % (i, group*3+1, n), img[y:y+h, x:x+w])
                n += 1
            """ id """
            if not os.path.isdir('Segmentation/%d/%d' % (i, group*3+2)):
                os.mkdir('Segmentation/%d/%d' % (i, group*3+2))
            n = 0
            for contour in sorted(contours[group*37+19:group*37+37], key=lambda contour: cv.minEnclosingCircle(contour)[0][0]):
                [x, y, w, h] = cv.boundingRect(contour)
                cv.imwrite('Segmentation/%d/%d/%d.jpg' % (i, group*3+2, n), img[y:y+h, x:x+w])
                n += 1
            group += 1
        for contour in contours:
            [x, y, w, h] = cv.boundingRect(contour)
            cv.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        """ save """
        cv.imwrite('Segmentation/%d.jpg' % i, img)
```

这一步中，我们使用 Python3.5 进行编程。我们使用 opencv 的 cv.findContours() 函数来寻找连通域。我们把连通域大小大于 96 的连通域视为字符，这里的 96 通过输入图像的分辨率计算得到。考虑到不同人的书写习惯不同，有部分输入字符是不连续的，我们先对二值化图像进行反色操作，然后通过膨胀操作把断裂的字符连接在一起。在输入图像中，我们可以发现不少字符行是不平行的，因此，我们很难通过 y 方向切割获得某一行的数据。为了简化操作，我们假设已知字符是按学号 - 手机号 - 身份证号的顺序书写的，且它们的长度分别为8、11、18。那么我们只需依次从左到右读入最上方的 8 个字符、中间的 11 个字符、最下方的 18 个字符，即可获得每一行的字符。我们使用 cv.boundingRect() 函数把每个切割好的字符提取出来，并保存到 Segmentation 文件夹中。其中，Segmentation 根目录下保存的是图



像的整体切割结果，Segmentation/sn 保存的是第 sn 幅图每一行的切割结果。



这里给出其中一个样例，对应输入图像 0.jpg。

4. 针对单个切割好的字符，进行分类识别。

我们在上一次实验中已经尝试了使用 Adaboost 来进行分类，但是优化过后仍仅能在 Mnist 测试集上获得 0.8847 的准确率。如果使用 Adaboost 模型，我们显然无法在中间的数据集上获得较好的结果，因此，我尝试了 NN 与 CNN 两种不同的模型，并在 CNN 上取得较好的效果。

```
def load_mnist(path, kind='train'):
    """ load MNIST data from 'path' """
    label_path = os.path.join(path, '%s-labels.idx1-ubyte' % kind)
    image_path = os.path.join(path, '%s-images.idx3-ubyte' % kind)

    with open(label_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II', lbpath.read(8))
        labels = np.fromfile(lbpath, dtype=np.uint8)

    with open(image_path, 'rb') as imgpath:
        magic, n, rows, cols = struct.unpack('>IIII', imgpath.read(16))
        images = np.fromfile(imgpath, dtype=np.uint8).reshape(len(labels), 784)

    return images, labels
```

我们先使用 load_mnist() 函数读入 Mnist 数据集。

```
def binary(x):
    images = np.reshape(x, (-1, 28, 28))
    for i in range(images.shape[0]):
        images[i] = cv.threshold(images[i], 63, 255, cv.THRESH_BINARY)[1]
    return np.reshape(images, (-1, 784))
```

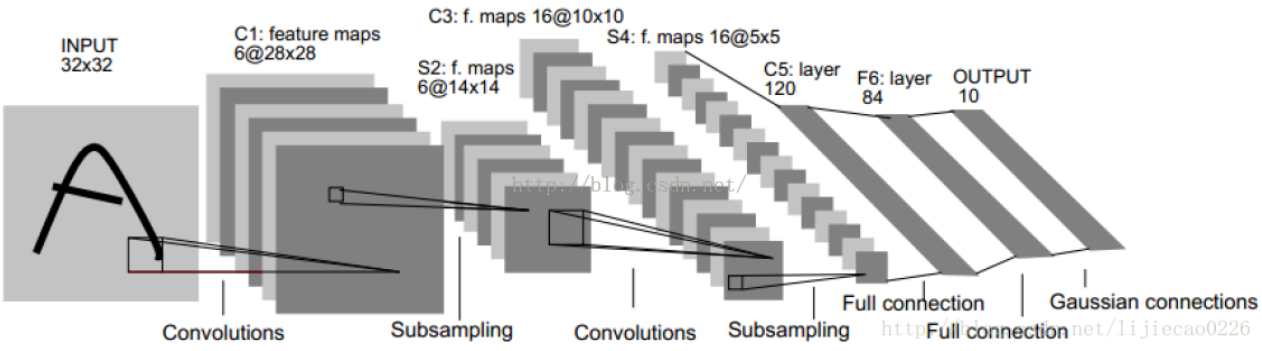
尽管 Mnist 数据集都是灰度图，我们在正式训练前，需要先对 Mnist 数据集进行二值化。

```
x_train, y_train = load_mnist('Mnist')
x_test, y_test = load_mnist('Mnist', 't10k')
x_train = binary(x_train)
x_test = binary(x_test)
""" parameters """
alpha = 1.0
epochs = 25
batch_size = 100
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])
dataset = tf.data.Dataset.from_tensor_slices((x, y)).shuffle(20).batch(batch_size).repeat()
""" weight, bias """
w1 = tf.Variable(tf.random_normal([784, 392], stddev=0.03), name='w1')
b1 = tf.Variable(tf.random_normal([392]), name='b1')
w2 = tf.Variable(tf.random_normal([392, 10], stddev=0.03), name='w2')
b2 = tf.Variable(tf.random_normal([10]), name='b2')
""" hidden layer """
hidden_out = tf.nn.sigmoid(tf.add(tf.matmul(x, w1), b1))
""" output """
out = tf.nn.softmax(tf.add(tf.matmul(hidden_out, w2), b2))
""" loss """
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y * tf.log(tf.clip_by_value(out, 1e-10, 1.0)) +
                                              (1-y) * tf.log(tf.clip_by_value(1-out, 1e-10, 1.0)), axis=1))
optimizer = tf.train.AdadeltaOptimizer(learning_rate=alpha).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(out, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
""" initialization """
init = tf.global_variables_initializer()
iterator = dataset.make_initializable_iterator()
data_element = iterator.get_next()
""" one hot """
y_train_one_hot = tf.one_hot(y_train, 10, 1, 0)
y_test_one_hot = tf.one_hot(y_test, 10, 1, 0)
""" saver """
saver = tf.train.Saver(max_to_keep=1)
```

对于普通的 NN，我们使用 784 x 392 x 10 的网络来进行分类。我们使用 sigmoid 作为隐层的激活函数，使用普通的逻辑代价函数作为 cross entropy，使用自适应的 adam 作为优化器。为了防止过拟合，且加快训练速度，我们使用 shuffle 和 batch 来提升模型性能。值得注意的是，Mnist 的 label 是使用十进制数字储存的，因此，我们还需要在训练前对类标进行 one hot 编码。为了能重复使用训练好的模型，我们使用 saver 来保存训练好的模型。

```
""" session """
with tf.Session() as sess:
    sess.run(init)
    y_train = sess.run(y_train_one_hot)
    y_test = sess.run(y_test_one_hot)
    sess.run(iterator.initializer, feed_dict={x: x_train, y: y_train})
    total_batch = len(y_train) // batch_size
    for epoch in range(epochs):
        for step in range(total_batch):
            x_batch, y_batch = sess.run(data_element)
            res, c = sess.run([optimizer, cross_entropy], feed_dict={x: x_batch, y: y_batch})
        print("Epoch:", (epoch + 1), "accuracy =", sess.run(accuracy, feed_dict={x: x_test, y: y_test}))
    saver.save(sess, 'Model/mnist')
```

我们使用 tensorflow session 来运行构建好的模型，并在每一个 epoch 输出模型在 Mnist 测试集上的表现。最后，我们能获得 97% 的准确率。但是，由于我们处理的是图片，我们使用 NN 依然无法在自己的数据集中取得较好的分类结果。因此，我们将详细介绍 CNN 的做法，NN 部分的识别结果可在上一次的 part1 的提交中看到，在 part2 中仅提供 CNN 的识别结果，其中，Result 文件夹中的 stage1 和 stage2 分别为两个阶段的识别结果。



我们使用的卷积网络结构如上图所示。

```
x_train, y_train = load_mnist('Mnist')
x_test, y_test = load_mnist('Mnist', 't10k')
x_train = binary(x_train)
x_test = binary(x_test)
""" parameters """
alpha = 1.0
epochs = 25
batch_size = 100
x = tf.placeholder(tf.float32, [None, 784], name='x')
y = tf.placeholder(tf.float32, [None, 10], name='y')
dataset = tf.data.Dataset.from_tensor_slices((x, y)).shuffle(20).batch(batch_size).repeat()
""" weight, bias """
w_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 6], stddev=0.1), name='w_conv1')
b_conv1 = tf.Variable(tf.random_normal([6]), name='b_conv1')
w_conv2 = tf.Variable(tf.truncated_normal([5, 5, 6, 16], stddev=0.1), name='w_conv2')
b_conv2 = tf.Variable(tf.random_normal([16]), name='b_conv2')
w_fc1 = tf.Variable(tf.truncated_normal([7*7*16, 120], stddev=0.1), name='w_fc1')
b_fc1 = tf.Variable(tf.random_normal([120]), name='b_fc1')
w_fc2 = tf.Variable(tf.truncated_normal([120, 10], stddev=0.1), name='w_fc2')
b_fc2 = tf.Variable(tf.random_normal([10]), name='b_fc2')
""" hidden layer """
h_conv1 = tf.nn.relu(tf.nn.conv2d(tf.reshape(x, [-1, 28, 28, 1]), w_conv1, strides=[1, 1, 1, 1], padding='SAME')
                     + b_conv1)
h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
h_conv2 = tf.nn.relu(tf.nn.conv2d(h_pool1, w_conv2, strides=[1, 1, 1, 1], padding='SAME') + b_conv2)
h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
h_fc1 = tf.nn.relu(tf.matmul(tf.reshape(h_pool2, [-1, 7*7*16]), w_fc1) + b_fc1)
""" output """
keep_prob = tf.placeholder(tf.float32, name='keep_prob')
y_conv = tf.nn.softmax(tf.matmul(tf.nn.dropout(h_fc1, keep_prob), w_fc2) + b_fc2, name='y_conv')
""" loss """
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y * tf.log(tf.clip_by_value(y_conv, 1e-10, 1.0)) +
                                              (1-y) * tf.log(tf.clip_by_value(1-y_conv, 1e-10, 1.0)), axis=1))
optimizer = tf.train.AdadeltaOptimizer(learning_rate=alpha).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_conv, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name='accuracy')
""" initialization """
init = tf.global_variables_initializer()
iterator = dataset.make_initializable_iterator()
data_element = iterator.get_next()
""" one hot """
y_train_one_hot = tf.one_hot(y_train, 10, 1, 0)
y_test_one_hot = tf.one_hot(y_test, 10, 1, 0)
""" saver """
saver = tf.train.Saver(max_to_keep=1)
```

通过卷积操作，我们可以对特征进行浓缩，增强感受野，从而提升训练效果。接下来，我们使用池化层对图像进行压缩，并最后使用全连接层来输出分类结果。这里，除了使用了 shuffle 和 batch 的技巧外，我们还额外使用了 dropout 来防止过拟合，此处的 dropout 比例为 50%。其余操作大体与 NN 相同，但我们还是要注意使用 tf.clip_by_value() 来防止 log 函数的输入为无效值，从而导致优化器无法正常优化代价。

```
""" session """
with tf.Session() as sess:
    sess.run(init)
    y_train = sess.run(y_train_one_hot)
    y_test = sess.run(y_test_one_hot)
    sess.run(iterator.initializer, feed_dict={x: x_train, y: y_train})
    total_batch = len(y_train) // batch_size
    for epoch in range(epochs):
        for step in range(total_batch):
            x_batch, y_batch = sess.run(data_element)
            res, c = sess.run([optimizer, cross_entropy], feed_dict={x: x_batch, y: y_batch, keep_prob: 0.5})
        print("Epoch:", (epoch + 1), "accuracy =",
              sess.run(accuracy, feed_dict={x: x_test, y: y_test, keep_prob: 1.0}))
    saver.save(sess, 'Model/mnist')
```

在训练好模型后，我们读取模型并对之前提取好的数字进行分类。

```
def clf():
    if not os.path.isfile('Model/checkpoint'):
        train_cnn()
    graph = tf.get_default_graph()
    """ session """
    with tf.Session(graph=graph) as sess:
        """ loader """
        loader = tf.train.import_meta_graph('Model/mnist.meta')
        loader.restore(sess, tf.train.latest_checkpoint('Model'))
        """ tensor """
        x = graph.get_tensor_by_name('x:0')
        keep_prob = graph.get_tensor_by_name('keep_prob:0')
        out = graph.get_tensor_by_name('y_conv:0')
        """ get the output of different cases """
        res = open('Result/stage2.csv', mode='r')
        lines = res.readlines()
        res.close()
        res = open('Result/stage2.csv', mode='w')
        res.write(lines[0])
        lines.pop(0)
        for i in range(0, 63):
            res.write(lines[0][:-1])
            lines.pop(0)
            row = 0
            while os.path.isdir('Segmentation/%d/%d' % (i, row)):
                col = 0
                digits = ''
                while os.path.isfile('Segmentation/%d/%d/%d.jpg' % (i, row, col)):
                    img = cv.imread('Segmentation/%d/%d/%d.jpg' % (i, row, col))
                    grey = 255 - cv.cvtColor(img, cv.COLOR_BGR2GRAY)
                    if grey.shape[0] > grey.shape[1]:
                        [height, width] = [20, int(20*grey.shape[1]/grey.shape[0])]
                        scale = cv.copyMakeBorder(cv.resize(grey, (width, height)), 4, 4, 14-int(width/2),
                                                  14-width+int(width/2), cv.BORDER_CONSTANT, value=0)
                    else:
                        [height, width] = [int(20*grey.shape[0]/grey.shape[1]), 20]
                        scale = cv.copyMakeBorder(cv.resize(grey, (width, height)), 14-int(height/2),
                                                  14-height+int(height/2), 4, 4, cv.BORDER_CONSTANT, value=0)
                    scale = cv.threshold(scale, 63, 255, cv.THRESH_BINARY)[1]
                    # cv.imwrite('%d.jpg' % col, scale)
                    digits += str(sess.run(tf.argmax(out, 1), feed_dict={x: np.reshape(scale, (1, 784)), keep_prob: 1.0})[0])
                    col += 1
                res.write(digits + ',')
                row += 1
            res.write('\n')
        res.close()
```

由于我们的输入并不是 28x28 的图像，我们需要先使用 cv.copyMakeBorder() 来调整其分辨率。但是，如果我们使用非同性放缩的话，数字的形状将会改变。因此，我们先获取图像的长边，并基于长边进行同性放缩，之后使用 padding 的方法将图像调整为 28x28 的大小。由于我们进行了放缩操作，我们需要重新对图像进行二值化，此时，我们只需进行简单的全局阈值二值化即可。我们通过运行 tf.argmax(out, 1) 即可获得分类结果。

通过对比使用 NN 获得的 stage1 结果（已在 part1 提交中上传，此次不再重复上传）与使用 CNN 获得的 stage1 结果，我们能发现分类器能更好地处理 5、7 等数字了，识别的整体准确率提升了不少。

最后，我们把 stage1 的结果保存在 Stage1.zip 中，stage2 的结果保存在根目录中。