

第一周-图像识别-PRML 小组项目报告

组员：叶帆帆（浙江大学）、梁宵（浙江大学）

项目成绩




		76-叶帆帆-PRML	0.944022	1	18-02-03 22:03	18-02-03 22:03
12		74-梁宵-PRML	0.907723	10	18-02-04 16:13	18-02-04 19:32

图 1 项目成绩 (NO.2 & NO.12)

项目内容及目的

本周主要对手写汉字进行识别分类，旨在锻炼深度学习构建网络，处理数据，进行训练与测试的实战能力。我们同时使用 Caffe 和 TensorFlow 框架进行项目实战。Caffe 使用的是略微修改的 Alexnet，TensorFlow 使用的是自己设计的一个网络。下面我们将分别介绍两种框架下的具体实现过程。

项目测试环境及数据来源

系统环境：科赛 K-Lab&线下服务器（GPU：1080Ti）

数据来源：科赛平台提供的数据+中科院自动化所提供的补充数据

http://www.nlpr.ia.ac.cn/databases/handwriting/Application_form.html

实战过程

● 基于 caffe 框架的实现过程（线下初步训练，线上测试）

数据预处理

① 给数据集打标签

我拿到任务后第一反应是先了解下数据集的形式，看完数据集以后非常感动。科赛给我们提供的数据集竟然是已经分好类的图片形式的数据集，这给作为深度学习新手的我节约不少时间。于是我就愉快的给数据集打起了标签，几行代码就轻松搞定。

```

1 src = '/mnt/datasets/train/'
2 fileNameList = os.listdir(src)
3 for fileName in fileNameList:
4     with open('/mnt/datasets/train/train.txt', 'a') as f:
5         for imgName in os.listdir(src+fileName):
6             f.write(fileName+'/' +imgName+' '+str(int(fileName))+'\n')
7 f.close()

```

图 2 打标签代码截图

② 将图片数据及转化成 leveldb 格式的数据集

Caffe 提供非常多适用的工具和接口，在处理完数据的标签之后，将 train 和 validation 的数据转化成 leveldb 格式的数据集，这一步只需要调用 caffe 提供的接口函数即可，具体如下：

```

convert.sh (~/Desktop/LX/handwrite/data) - gedit
1 DATA=/home/ssw/Desktop/LX/handwrite/data
2 rm -rf $DATA/valleveldb
3 /home/ssw/caffe/build/tools/convert_imageset --backend=leveldb --shuffle=true --
  resize_height=108 --resize_width=108 /home/ssw/Desktop/LX/handwrite/data/test/ $DATA/test.txt
  $DATA/valleveldb|

```

图 3 数据转化脚本

转化的过程中将数据 resize 为 108*108（网络经验值）。值得一提的是，在制作 leveldb 的时候混洗（shuffle）一定要打开，不然很可能不收敛。

网络结构选择

根据经验结合图像大小和内容复杂度，我和我的队友一致认为不需要很深的网络应该就能比较好的解决该问题。由于提供的数据集图片大小较小，故也不能再网络中加太多的 pooling 层，几番思考后，我决定用 AlexNet。网络结构图如下：

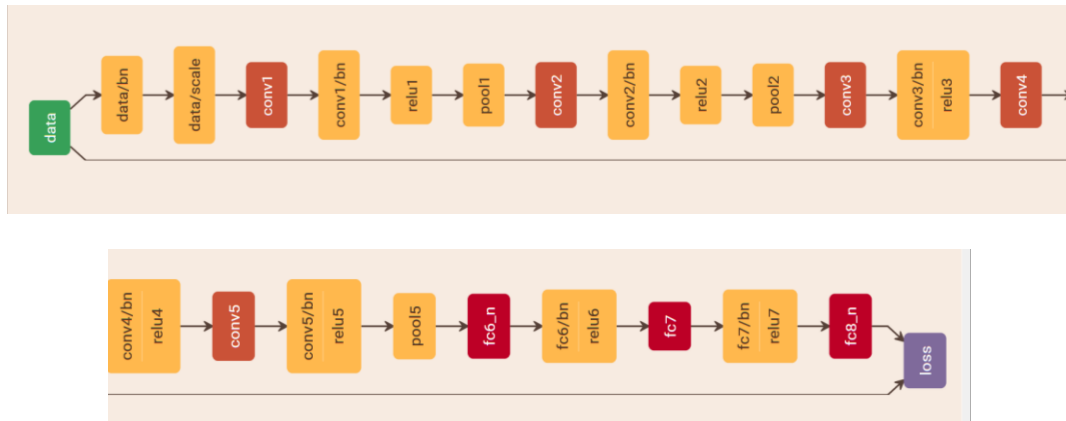


图 4 AlexNet 网络结构

Loss 函数选择

对于多分类问题，loss 函数就非 SoftMaxWithLoss 莫属了，其核心公式如下：

$$x_i = x_i - \max(x_1, \dots, x_n)$$

$$p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

$$Loss = -\log p_k \quad \text{样本Label为} k$$

caffe 中也提供了 SoftMaxWithLoss 层的实现，可以说是很方便了。

训练参数详细

这一部分是最关键也是最费时的。首先我采用最简单的 SGD 优化器，而且没有使用预训练模型，而且没有 shuffle 我的训练数据集。结果可想而知，网络训练了一整夜，也没有收敛。后来我查阅相关文献后，将优化器改为 Adam，重新制作 shuffle 后的数据集，也加入了预训练模型，果然很快收敛，但是在 loss=1 附件来回震荡，验证准确率也在 70%左右徘徊。后来又将学习速率 lr 减小，重新训练。经过数十个小时的训练后，得到了还不错的结果，验证准确率到了 95%左右，loss 也到了 0.03 左右。详细的参数设置如下：

```
test_iter: 3500
test_interval: 1000
base_lr: 0.0005
display: 200
max_iter: 250000
lr_policy: "poly"
power: 1
momentum: 0.9
weight_decay: 0.0005
snapshot: 2000
snapshot_prefix: "model_iter_"
random_seed: 0
net: "/home/ssw/Desktop/LX/handwrite/data/train_val.prototxt"
test_initialization: false
iter_size: 2
type: "Adam"
stepsize: 3000
solver_mode: GPU
```

图 5 训练参数详细

训练测试结果

根据 caffe 训练的日志文件，绘制训练 loss 和 accuracy 图如下：

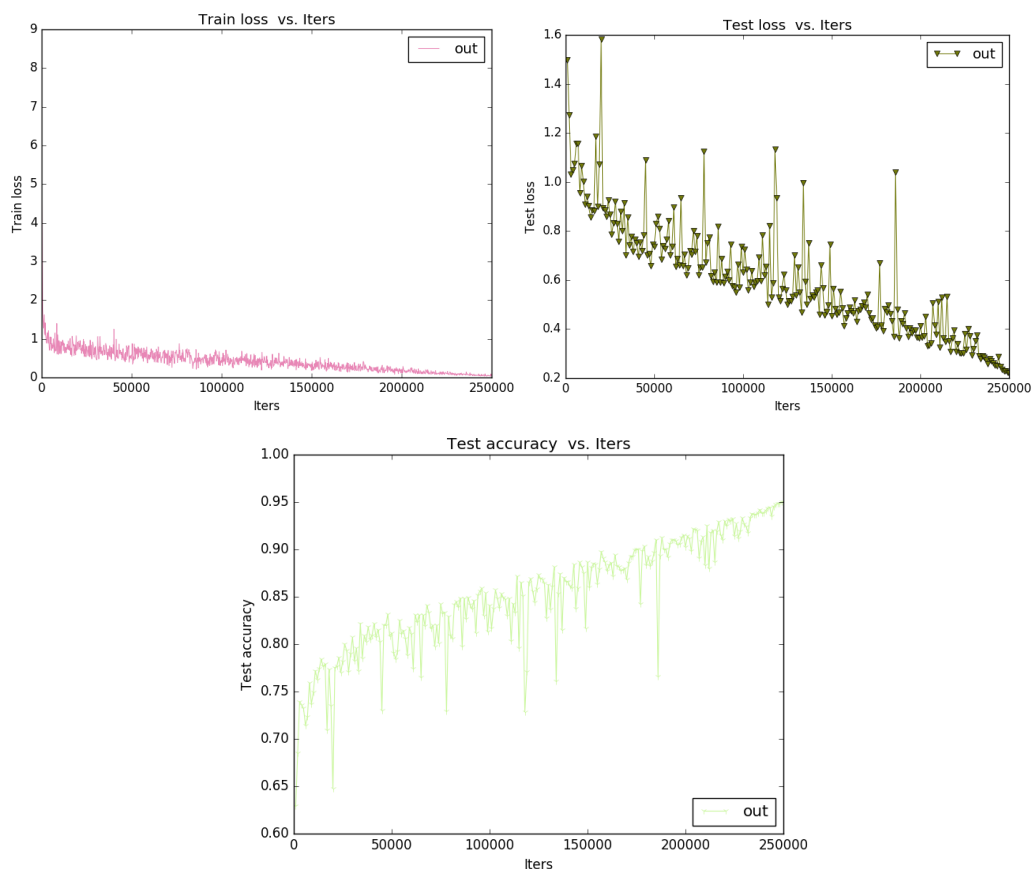


图 6 分别为训练 loss、测试 loss 和验证准确度图

线上结果测试

线上一共提供了 39000+个测试集，对测试集中的输入图片逐一测试，并生成 json 格式的结果输入，具体代码如下：

根据训练（迭代250000次）结果参数文件，对test测试输出结果如下

```
In [4]: import matplotlib.pyplot as plt
import os
import caffe
import numpy as np
import pickle
import json
import tqdm
root="/home/kesci/work/" #根目录
deploy=root + 'deploy.prototxt' #deploy文件
caffe_model=root + 'model_iter_250000.caffemodel' #训练好的 caffemodel
net = caffe.Net(deploy,caffe_model,caffe.TEST) #加载model和network

#图片预处理设置
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape}) #设定图片的shape格式(1,3,28,28)
transformer.set_transpose('data', (2,0,1)) #改变维度的顺序，由原始图片(28,28,3)变为(3,28,28)
transformer.set_mean('data', np.load(mean_file).mean(1).mean(1)) #减去均值，前面训练模型时没有减均值，这儿就不用
transformer.set_raw_scale('data', 1) # 缩放到【0, 255】之间
transformer.set_channel_swap('data', (2,1,0)) #交换通道，将图片由RGB变为BGR
#加载label字典，线上不需要，现在需要
char_dict = pickle.load(open('./char_dict', 'rb'))
#字典键值对换序
src = '/mnt/datasets/test/'
submit_name = '/home/kesci/work/submit.json'
fileName = os.listdir(src)
answer=[]
for idx,imgName in enumerate(fileName):
    img = src+imgName
    im = caffe.io.load_image(img) #加载图片
    net.blobs['data'].data[...] = transformer.preprocess('data', im) # 执行上面设置的图片预处理操作，并将图片载入到blob中
    out = net.forward()
    top_k = net.blobs['prob'].data[0].flatten().argmax()
    save_dict = dict()
    save_dict['label'] = int(top_k) #需要加int转化
    save_dict['filename'] = imgName
    r = json.dumps(save_dict)
    answer.append(r)
    print(str(idx)+'done')
with open(submit_name,'w') as f:
    wirte_str = '['+','.join(answer)+']'
    f.write(wirte_str)
print("finish!")
```

图 7 测试代码截图

最后线上的结果是 94.4022%，第二名，是一个还可以的结果。由于时间紧迫也没有再做进一步提升，这是我感到遗憾的一点，希望下周能够继续加油。在此再次向志愿者和辅导员以及相关工作人员表示感谢。

● 基于 TensorFlow 框架的实现过程

数据预处理

读取路径后，对数据进行混洗，防止过拟合，并按文件夹名对数据进行标记。

```
def get_image_path_and_labels(dir):
    img_path = []
    for root, dir, files in os.walk(dir):
        img_path += [os.path.join(root, f) for f in files]
    # Shuffle数据
    random.shuffle(img_path)
    labels = [int(name.split(os.sep)[-2]) for name in img_path]
    return img_path, labels
```

图 8 TensorFlow 数据处理代码

由于数据集很大，对数据进行分批操作，对图片进行 resize，大小我设为 96*96，将图片与 label 对应好后返回。

```

def batch(dir, batch_size, preprocess=False):
    img_path, labels = get_image_path_and_labels(dir)
    # 将数据转为tensor
    img_tensor = tf.convert_to_tensor(img_path, dtype=tf.string)
    lb_tensor = tf.convert_to_tensor(labels, dtype=tf.int64)
    # 分批读取图片
    input_pipe = tf.train.slice_input_producer([img_tensor, lb_tensor])
    # 把图片转为灰度图
    img = tf.read_file(input_pipe[0])
    imgs = tf.image.convert_image_dtype(tf.image.decode_png(img, channels=1), tf.float32)
    # 随机修改图片避免过拟合
    if preprocess:
        imgs = tf.image.random_contrast(imgs, 0.9, 1.1)
    # resize图片
    imgs = tf.image.resize_images(imgs, tf.constant([FLAGS.img_size, FLAGS.img_size], dtype=tf.int32))
    # 读取label
    lbs = input_pipe[1]
    img_batch, lb_batch = tf.train.shuffle_batch([imgs, lbs], batch_size=batch_size, capacity=50000,
                                                min_after_dequeue=10000)
    return img_batch, lb_batch

```

图 9 TensorFlow 数据处理代码 2

搭建网络

考虑到上传时模型不能大于 100m，同时图片 resize 后较小，搭建了一个较浅的网络，由 5 个卷积层和两个全连接层构成。

第一层：5*5 的卷积核 32 个，步长为 1；

第二层：3*3*32 的卷积核 64 个，步长为 1；

第三层：3*3*64 的卷积核 128 个，步长为 1；

第四层：3*3*128 的卷积核 256 个，步长为 1；

第五层：3*3*256 的卷积核 512 个，步长为 1；

其中每层经过一个 relu 层和 2*2 的 maxpooling 层

最后是两个全连接层。

```

conv1 = slim.conv2d(img, 32, [5, 5], 1, padding="SAME", scope="conv1")
relu1 = tf.nn.relu(conv1)
pool1 = slim.max_pool2d(conv1, [2, 2], [2, 2], padding="SAME")
conv2 = slim.conv2d(pool1, 64, [3, 3], padding="SAME", scope="conv2")
relu2 = tf.nn.relu(conv2)
pool2 = slim.max_pool2d(conv2, [2, 2], [2, 2], padding="SAME")
conv3 = slim.conv2d(pool2, 128, [3, 3], padding="SAME", scope="conv3")
relu3 = tf.nn.relu(conv3)
pool3 = slim.max_pool2d(conv3, [2, 2], [2, 2], padding="SAME")
conv4 = slim.conv2d(pool3, 256, [3, 3], [2, 2], scope="conv4", padding="SAME")
relu4 = tf.nn.relu(conv4)
pool4 = slim.max_pool2d(conv4, [2, 2], [2, 2], padding="SAME")
conv5 = slim.conv2d(pool4, 512, [3, 3], [2, 2], scope="conv5", padding="SAME")
relu5 = tf.nn.relu(conv5)
pool5 = slim.max_pool2d(conv5, [2, 2], [2, 2], padding="SAME")
# flat feature map, 以便我们可以将其连接到完全连接的图层
flat = slim.flatten(pool5)
# 两个全连接层
fcnet1 = slim.fully_connected(slim.dropout(flat, keep_prob=keep_prob), 1024, activation_fn=tf.nn.tanh, scope="fcnet1")
fcnet2 = slim.fully_connected(slim.dropout(fcnet1, keep_prob=keep_prob), 3755, activation_fn=None, scope="fcnet2")
loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=fcnet2, labels=labels))
# 将结果与真实label对比获得准确率
accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(fcnet2, 1), labels), tf.float32))
step = tf.get_variable("step", shape=[], initializer=tf.constant_initializer(0), trainable=False)

```

图 10 TensorFlow 模型构造代码

Tensorflow 中的 graph 图如下图所示：

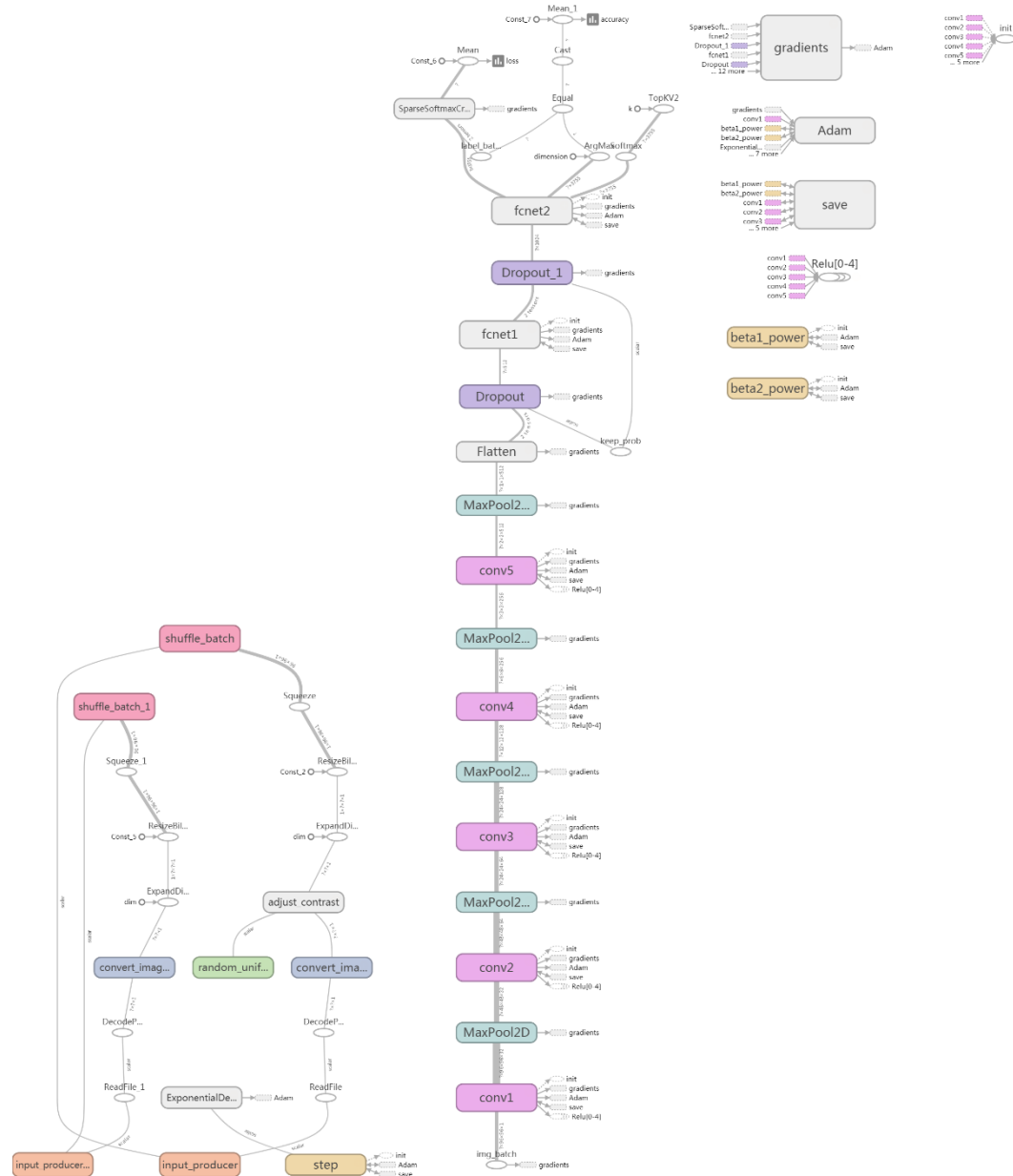


图 11 TensorFlow 模型 graph 图

优化算法和参数设置

使用 Adam 优化算法，初始学习率为 $2e-4$, $decay_rate=0.97$, $decay_steps=2000$, 指数部分采用整除策略，时间有限，具体参数还有待调整。

```
lrate = tf.train.exponential_decay(2e-4, step, decay_rate=0.97, decay_steps=2000, staircase=True)
# 使用Adam优化算法
optimizer = tf.train.AdamOptimizer(learning_rate=lrate).minimize(loss, global_step=step)
prob_dist = tf.nn.softmax(fcnet2)
```

图 12 TensorFlow 参数设置

训练

进行了 20w 步，每 1w 步保存一个模型，每 500 步打印一次验证集的准确度，并输出 log 文件。

loss

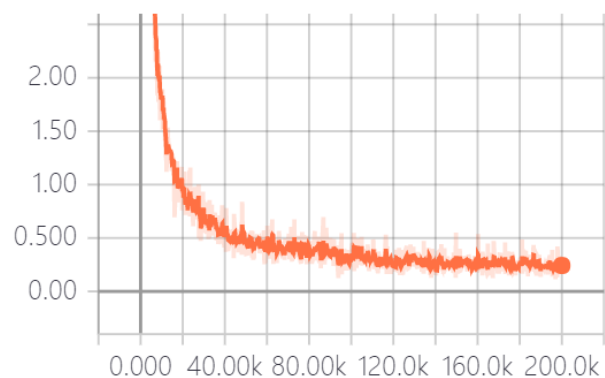


图 13TensorFlow 训练 Loss

accuracy

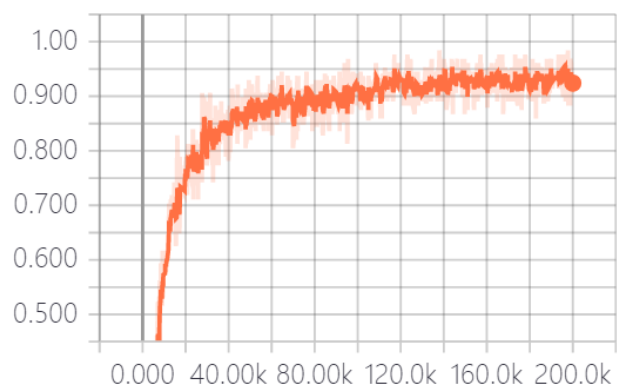


图 14TensorFlow 训练准确率

可以看出 loss 在 0.5 附近不断震荡，准确度也在 0.9 附近震荡。由于时间有限，没有进行更多的尝试，我认为降低初始学习率也许能解决这个问题，因为在接近 5w 步的时候 loss 就一直在震荡了。

测试

测试使用的是平台的测试图片，采用的办法是一张张读图片进行 label 的输出，然后写入 json 文件。

```
def test(path):
    # Read test picture and resize it, turn it to grey scale.
    img_path = list()
    filename = list()
    label = list()
    save_dict = dict()
    result = []
    graph = cnn()
    with tf.Session() as sess:
        saver = tf.train.Saver()
        saver.restore(sess=sess, save_path=tf.train.latest_checkpoint(FLAGS.checkpoint))
        for root,dir, files in os.walk(path):
            img_path= [os.path.join(root, f) for f in files]
            filename = [os.path.join(f) for f in files]
            for i in range(len(img_path)):
                tst_image = cv2.imread(img_path[i],cv2.IMREAD_GRAYSCALE)
                tst_image = cv2.resize(tst_image,(96, 96))
                tst_image = numpy.asarray(tst_image) / 255.0
                tst_image = tst_image.reshape([-1, 96, 96, 1])
                # 将图片读入网络
                graph_dict = {graph['img']: tst_image, graph['keep_prob']: 1.0}
                val, index = sess.run([graph['val_top3'], graph['index_top3']], feed_dict=graph_dict)
                #print("Probability: %.5f"%val[0][0]+ " with label:"+str(index[0][0]),i-1)
                label.append(str(index[0][0]))
                #img_path[i-1]=FLAGS.train_dir+"/"+ "%0.5d" % index[0][0]
                # select one of the picture from the label with top 1 probability
                save_dict['label'] = int(label[i])
                save_dict['filename'] = filename[i]

            r = json.dumps(save_dict)
            result.append(r)
        #print(filename)
        #写入json
        with open('/home/kesci/work/submit.json','w') as f:
            write_str = '['+','.join(result)+']'
            f.write(write_str)
    return label
```

图 15 TensorFlow 测试代码

得到了如下所示的 json 文件：

```
!cat /home/kesci/work/submit.json
```

```
[{"label": 261, "filename": "6666.png"}, {"label": 1271, "filename": "2240.png"}, {"label": 942, "filename": "12569.png"}, {"label": 645, "filename": "21871.png"}, {"label": 1694, "filename": "22787.png"}, {"label": 1102, "filename": "21914.png"}, {"label": 1101, "filename": "13003.png"}, {"label": 1834, "filename": "34400.png"}, {"label": 1787, "filename": "1944.png"}, {"label": 376, "filename": "18977.png"}, {"label": 1441, "filename": "18968.png"}, {"label": 1091, "filename": "22589.png"}, {"label": 1179, "filename": "31336.png"}, {"label": 1241, "filename": "18713.png"}, {"label": 595, "filename": "31639.png"}, {"label": 1061, "filename": "11704.png"}, {"label": 1593, "filename": "6060.png"}, {"label": 1664, "filename": "25227.png"}, {"label": 1402, "filename": "29988.png"}, {"label": 670, "filename": "30242.png"}, {"label": 1724, "filename": "25321.png"}, {"label": 698, "filename": "1331.png"}, {"label": 1495, "filename": "28849.png"}, {"label": 145, "filename": "16203.png"}, {"label": 1495, "filename": "971.png"}, {"label": 544, "filename": "24248.png"}, {"label": 1127, "filename": "14796.png"}, {"label": 1448, "filename": "4125.png"}, {"label": 1081, "filename": "19315.png"}, {"label": 1453, "filename": "16880.png"}, {"label": 319, "filename": "16466.png"}, {"label": 230, "filename": "14990.png"}, {"label": 388, "filename": "9313.png"}, {"label": 1370, "filename": "17360.png"}, {"label": 1407, "filename": "31018.png"}, {"label": 1154, "filename": "26967.png"}, {"label": 1585, "filename": "14940.png"}, {"label": 1800, "filename": "19487.png"}, {"label": 1182, "filename": "7908.png"}, {"label": 725, "filename": "31755.png"}, {"label": 919, "filename": "35988.png"}, {"label": 553, "filename": "29997.png"}, {"label": 979, "filename": "27565.png"}, {"label": 1586, "filename": "1983.png"}, {"label": 650, "filename": "11754.png"}, {"label": 1594, "filename": "18099.png"}, {"label": 1766, "filename": "14523.png"}, {"label": 315, "filename": "27490.png"}, {"label": 1831, "filename": "9777.png"}, {"label": 1606, "filename": "18244.png"}, {"label": 1502, "filename": "20609.png"}, {"label": 863, "filename": "4868.png"}, {"label": 255, "filename": "2339.png"}, {"label": 1622, "filename": "599.png"}, {"label": 868, "filename": "13716.png"}, {"label": 502, "file
```

图 16 结果截图

将其提交，准确率在 90.7%，不算太理想。

由于 resize 后图片大小与 Alexnet 的理想输入大小差的较大，再加上本身对 tensorflow 的不熟悉，没有用 tf 搭建 Alexnet，在这个网络跑完后我们又尝试了个更浅的网络，过程与结果都大同小异。

遇到的困难和思考

梁宵:在使用 TensorFlow 的过程中遇到了不少困难,因为之前我们使用的都是 Caffe 框架,没有接触过 TensorFlow,因此会碰到不少低级错误,在老师和同学们的帮助下解决了。最大的困难还是到最后 loss 一直震荡,无法下降,我觉得可以从网络的搭建和参数的设置上入手,网络的搭建可以参考我们 Caffe 框架使用的网络,参数设置上,我认为可以适当调低学习率。在这里也感谢一下队友,能从对方身上学到很多东西。

叶帆帆：看似简单的项目完整做完还是会碰到很多困难。我在第一周的项目中遇到的主要困难是网络对参数的调节，在项目过程中也积累了一些经验，希望下周能少犯一些低级错误。第一个小项目的顺利完成，得益于工作人员以及志愿者的热心服务，再次表示感谢。

希望和导师、同学交流的地方

希望能在群里多讨论一些代码实现方面的问题，这些都是较为薄弱的环节，有些时候讨论后能发现更加方便的实现办法，受益匪浅。