基于卷积神经网络的手写数字识别

序号	学号	专业班级	姓名	性别
1	**3140103367**	**软工 1401**	**杨奕辉**	**男**

1. Project Introduction

操作系统：Windows 10

语言框架：Python 3.5.2, Tensorflow-gpu

所用到的库：Numpy, sklearn, tensorflow.contrib

2. Technical Details

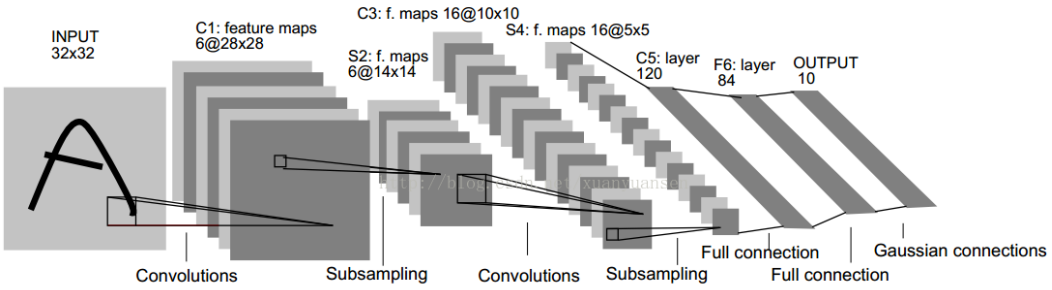


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet5 的实现过程：

1. Input 为 32*32 的图片。由于从数据集导入的图片为 28*28=784，故需要先对图片进行预处理。预处理函数实现如下：

```
x_image=tf.pad(tf.reshape(x,[-1,28,28,1]),[[0,0],[2,2],[2,2],[0,0]])
```

2. C1 层为卷积层。利用 6 个大小为 5*5 的卷积核，从 Input 数据中提取特征并执行卷积操作，得到 6 个大小为 28*28 的特征图。
3. S2 层为池化层。池化的 size 选择 2*2，这样将每个特征图压缩至 14*14 大小。

   我对卷积-池化作为一层操作，进行了封装：

```
def lenet5_layer(layer,weight,bias):
    W_conv=weight_variable(weight)
    b_conv=bias_variable(bias)

    h_conv=conv2d(layer,W_conv)+b_conv
    return max_pool_2x2(h_conv)
```

```
# convolution and pooling
def conv2d(x,W):
    return tf.nn.conv2d(x,W,strides=[1,1,1,1],padding='VALID')

def max_pool_2x2(x):
    return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='VALID')
```

Main 中的执行：

```
layer1=lenet5_layer(x_image,[5,5,1,6],[6])
```

```
# second layer
layer2=lenet5_layer(layer1,[5,5,6,16],[16])
```

4. C3 为卷积层。利用 16 个大小为 5*5 的卷积核，从 S2 层的每个特征图中提取数据并卷积，得到 16 个大小为 10*10 的特征图。
5. S4 为池化层。池化的 size 选择仍为 2*2，将每个特征图压缩至 5*5，得到 16 个 5*5 的特征图。
6. C5 为卷积层。利用 120 个大小为 5*5 的卷积核，从 S4 层的每个特征图中提取数据并卷积，得到 120 个大小为 1*1 的特征图。
   notice:由于该层卷积后的张量维度是[?, 1, 1, 120]，故需要手动 reshape。

```
# third layer
W_conv3=weight_variable([5,5,16,120])
b_conv3=bias_variable([120])

layer3=conv2d(layer2,W_conv3)+b_conv3
layer3=tf.reshape(layer3,[-1,120])
```

7. C5-F6-OUTPUT 均为全连接层。这里将密集连接层封装成函数，使用 full_connected 进行全连接。
   Notice：最后输出层的激活函数为 softmax，在输出之前使用了 dropout 来减少过拟合。

```
# connected layer
def dense_layer(layer,weight,bias):
    W_fc=weight_variable(weight)
    b_fc=bias_variable(bias)

    return tf.matmul(layer,W_fc)+b_fc
```

Main 中：

```
# all connected layer
con_layer1=dense_layer(layer3,[120,84],[84])

# output
con_layer2=dense_layer(con_layer1,[84,10],[10])
y_conv=tf.nn.softmax(tf.nn.dropout(y_conv),keep_prob)
```

8. 使用 tf.reduce_mean 和 softmax_cross_entropy_with_logits 来算交叉熵和训练准确率。Main 中如下：

```
# train and evalute
cross_entropy=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_,logits=y_conv))
train_step=tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction=tf.equal(tf.argmax(y_conv,1),tf.argmax(y_,1))
accuracy=tf.reduce_mean(tf.cast(correct_prediction,"float"))
```

这里 tf.argmax 的用途是找出当张量的所属类别。**由于 argmax 最终返回的是最大元素的 index，而 labels 也是由一个 1 和 9 个 0 组成的张量，故找出它们最大的元素的 Index，即可代表它们所属的类别。**

9. 对模型进行训练并测试，得出准确率。这里 batch_size=100，即每次训练 50 个样本数据，执行 30000 次迭代。其中有样本数据的重复使用。

```
for i in range(30000):
    batch=mnist.train.next_batch(50)
    if i%100==0:
        train_accuracy=accuracy.eval(feed_dict={
            x:batch[0],y_:batch[1],keep_prob:1.0
        })
        print("step %d,training accuracy %g"%(i,train_accuracy))
    train_step.run(feed_dict={x:batch[0],y_:batch[1],keep_prob:0.5})

print("Test accuracy %g"%accuracy.eval(feed_dict={
    x:mnist.test.images,y_:mnist.test.labels,keep_prob:1.0
}))
```

3. Experiment Results

1. 正常 LeNet5：

```
Test accuracy 0.9805
```

2. 增加或减少卷积层与池化层

- 在 S4 后增加一层卷积层，有 32 个卷积核，大小为 1x1：

```
Test accuracy 0.9733
```

- 在 S4 后增加一层卷积层与池化层，有 32 个卷积核，大小为 4x4。池化层此阿勇 max_pool。相应地，C5 的卷积核大小改为 1x1：

```
Test accuracy 0.9746
```

- 删除 C1，S2 层：

```
Test accuracy 0.9499
```

- 删除 C3，S4 层：

```
Test accuracy 0.9736
```

3. 改变卷积核大小

- Input->C1 层卷积核大小改为 9x9，S4->C5 层改为 4x4：

```
Test accuracy 0.9749
```

- Input->C1 层卷积核大小改为 9x9,S2->C3 层改为 7x7,S4->C5 层改为 3x3：

```
Test accuracy 0.9725
```

- S2->C3 层改为 7x7,S4->C5 层改为 4x4：

```
Test accuracy 0.7855
```

4. 改变卷积核数量

- +C1 层卷积核数量改为 14 个：

```
Test accuracy 0.8877
```

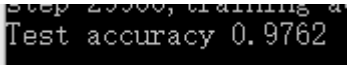
- +C1 层卷积核数量改为 10 个：

```
Test accuracy 0.9824
```

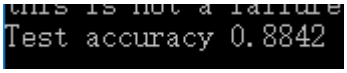
- -C1 层卷积核数量改为 2 个：

```
Test accuracy 0.9715
```

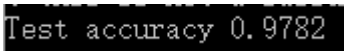
- -C3 层卷积核数量改为 10 个：



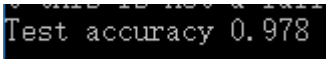
- +C3 层卷积核数量改为 24 个：



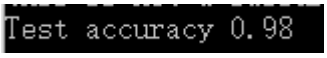
- -C5 层卷积核数量改为 80 个：



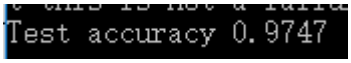
- +C5 层卷积核数量改为 140 个：



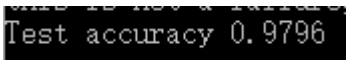
- ++C1 层卷积核数量改为 10 个，C3 层改为 20 个：



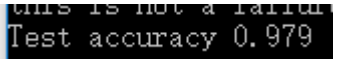
- +-C1 层卷积核数量改为 10 个，C3 层改为 12 个：



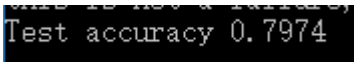
- +-C1 层卷积核数量改为 10 个，C5 层改为 80 个：



- ++C1 层卷积核数量改为 10 个，C5 层改为 140 个：

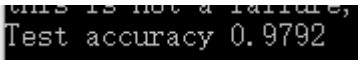


- ++C1 层卷积核数量改为 10 个，C5 层改为 160 个：

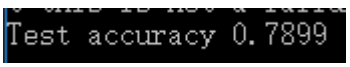


- +-C3 层卷积核数量改为 20 个，C5 层改为 80 个：

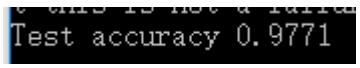
- ++C3 层卷积核数量改为 20 个，C5 层改为 160 个



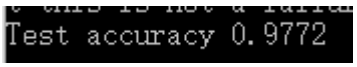
5. 改用 Relu 激活函数



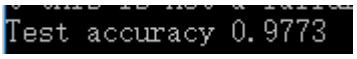
6. 改用 Tanh 激活函数



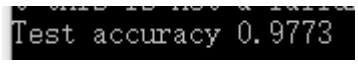
7. 改变全连接层大小
   - F6 层改为 64



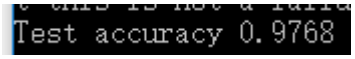
- F6 层改为 32



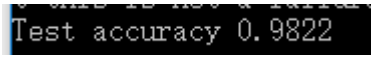
- F6 层改为 104



- F6 层改为 124



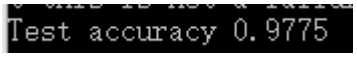
- F6 层改为 160



8. 增加一层全连接层

1. 在 C5 层后

- 增加一层神经元个数为 64 的全连接层

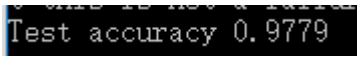


- 增加一层神经元个数为 100 的全连接层

- 增加一层神经元个数为 160 的全连接层

2. 在 F6 层后

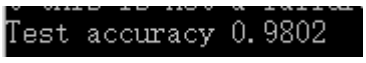
- 增加一层神经元个数为 64 的全连接层



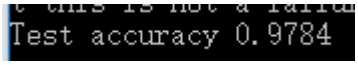
- 增加一层神经元个数为 32 的全连接层

- 增加一层神经元个数为 16 的全连接层

- 增加一层神经元个数为 102 的全连接层



- 增加一层神经元个数为 128 的全连接层



对比结果分析：

根据以上对比结果，我们不难看出，在 LeNet5 的基本网络结构和样本数据均不变的情况下，测试准确率基本变化不大，均在 97%~98%。

增加或减少卷积层和池化层、增加全连接层，均不能明显改变整个神经网络的性能，但会带来小幅度的准确率下降。

对于影响比较大的改变方案（上面已标红）,有以下几种情况：

1. 改变了 S2->C3 和 S4->C5 的卷积核大小。原因可能是 C3 层对 S2 的特征图的特征提取粒度不够细，导致提取到的特征比较复杂，从而在训练时有一定的过拟合，增大了泛化误差。

2. 改变卷积核数量：测试准确率的降低都发生在增加卷积核数量的实验用例中，原因可能是，当增加输出通道（增加特征图）后，实际是增加了特征神经元的复杂性，可以降低训练误差，但带来了一定的过拟合，导致泛化误差增大。

3. 改用 Relu 作为激活函数:好处是可以提高训练效率,但在本实验的过程中可以看到，改为 Relu 后,训练误差和泛化误差均增大了。原因可能是在 LeNet5 仅在输出层用激活函数，relu 会把小于 0 的数据直接归为 0 的类，因此带来了较大误差。

References:

[1]《lecun-98》Yann LeCun,Leon Bottou
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[2] tensorflow 官方网站 https://www.tensorflow.org/