浙江大學

本科实验报告

学习 CNN

课程名称：	计算机视觉
姓名：	夏豪诚
学院：	计算机科学与技术学院
专业：	信息安全
学号：	3170102492
指导老师：	宋明黎

2020 年 1 月 4 日

浙江大学实验报告

专业：信息安全
姓名：夏豪诚
学号：3170102492
日期：2020 年 1 月 4 日
地点：无

课程名称：计算机视觉　指导老师：宋明黎　成绩：
实验名称：学习 CNN　实验类型：综合实验　同组学生姓名：无

一、 实验环境

表 1: 测试环境

item	detail
CPU	Intel® Core™ i7-6700HQ CPU 2.60GHz
RAM	16.0GB DDR4 2133MHz
hard disk	SSD 256GB
OS	Windows 10 Pro 64-bit
TensorFlow	2.0.0
Python	Python 3.6.9 :: Anaconda, Inc.
Jupyter Notebook	6.0.2
opencv	4.1.2

二、 实验目的和要求

1. 实验目的

在 TensorFlow 框架下初步学习 CNN（卷积神经网络）。

2. 基本要求

利用 CNN 进行手写数字识别:
框架：TensorFlow （已包含下面网络结构与数据集）
数据集：The MnistDatabase of handwritten digits
网络结构：LeNet-5

1.1 具体任务

利用上述数据集、网络结构以及选定的 TensorFlow 框架实现手写数字的识别参考链接：

(1) MNIST 手写数字识别介绍 (已失效)

(2) MNIST 机器学习入门

(3) TensorFlow 从入门到精通（二）：MNIST 例程源码分析

三、 实验内容和步骤

1. 实验内容

(1) 获取 minist 数据集；

(2) 数据增强；

(3) 构建简单的 LeNet5 结构网络；

(4) 训练并检测手写数字识别结果正确性；

(5) 添加 dropout 和批量标准化重新训练。

针对这个部分的具体代码实现将在实验步骤中进行详细说明。

2. 实验步骤

3.2.0 实验环境配置

环境配置与之前实验相不同，所以我们可以从Anaconda官网上获得对应我们平台的 Anaconda 版本，并完成安装。

我们看到出现了 Anaconda Prompt。

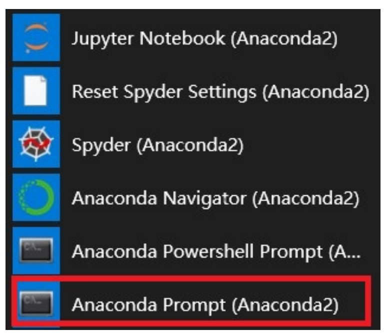

图 1: Anaconda Prompt

打开 Anaconda Prompt 创建环境，输入如下命令：

```
conda create -n tensorflow python=3.6 #创建配置名为tensorflow
activate tensorflow #激活
pip install tensorflow -i https://pypi.tuna.tsinghua.edu.cn/simple/
    #安装tensorflow（为了简便安装CPU版）
```

测试 tensorflow 是否安装成功，可以看到我们成功导入。

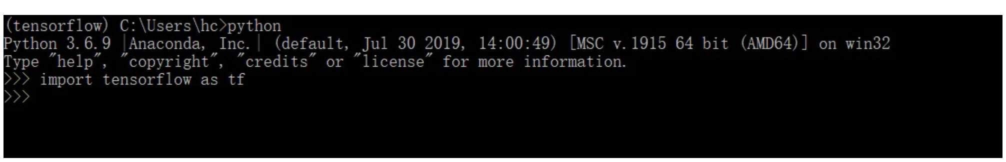

图 2: 导入 tensorflow

为了 jupyter 中使用此 tensorFlow 环境在激活 TensorFlow 之后安装 ipython 和 jupyter

```
conda install ipython
conda install jupyter
ipython kernelspec install-self --user #安装python kernel for Tensroflow:
pip install sklearn opencv-python matplotlib seaborn #安装依赖
```

在 menu 中可以看到出现了对应环境的 jupyter book，点开即可进入。

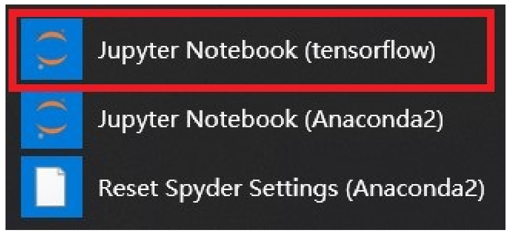

图 3: 使用 jupyter

在网页自动打开 Jupyter Notebook, 新建 Python3 Notebook 后进入如下界面。

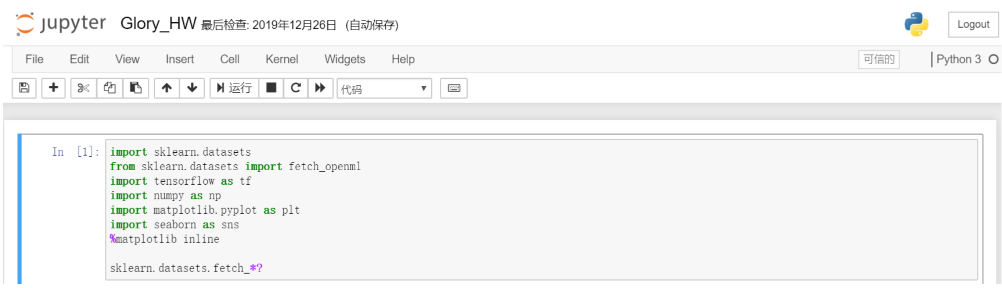

图 4: Jupyter Notebook 界面

3.2.1 获取 minist 数据集

在此处我通过 sklearn.datasets.fetch_openml 获取数据。

```
from sklearn.datasets import fetch_openml
# 查看可获取的内容 sklearn.datasets.fetch_*?

mnist = fetch_openml("mnist_784")

x = mnist["data"]
y = mnist["target"]
```

x 中为 60000 个手写数字图像 (图像尺寸为 $28 * 28$)，而 y 中为标记的对应的正确结果。

3.2.2 数据增强

这里我使用了两种方式来扩充数据集，一种是通过将图像向上、下、左、右分别移动一个像素，使得数据集扩大 4 倍；另一种则是水平翻转图像。

```
# shift ==> image & label
x_train_shifted = []
y_train_augmented = []

for dx, dy in ((1,0),(-1,0),(0,1),(0,-1)):
    for image, label in zip(x_train, y_train):
        x_train_shifted.append(shift_image(image, dx, dy))
        y_train_augmented.append(label)
x_train_shifted = np.array(x_train_shifted)
y_train_augmented = np.array(y_train_augmented)
x_train_shifted.shape, y_train_augmented.shape


# get the flipped img to enlarge the dataset
def horizontal_flip(images):
    flipped_images = []
    for img in images:
        flipped_img = cv2.flip(img, flipCode = 1)
        flipped_images.append(flipped_img)
    return (flipped_images)

flipped_imgs = horizontal_flip(x_train.reshape(-1, 28, 28))
flipped_imgs = np.array(flipped_imgs)
flipped_lables = np.array(y_train[:])
```

3.2.3 构建简单的 LeNet5 结构网络

由于输入图像尺寸大小为 $28*28$，而示例结构中的输入图像尺寸为 $32*32$, 这里仿照 LeNet-5 结构进行参数调整。

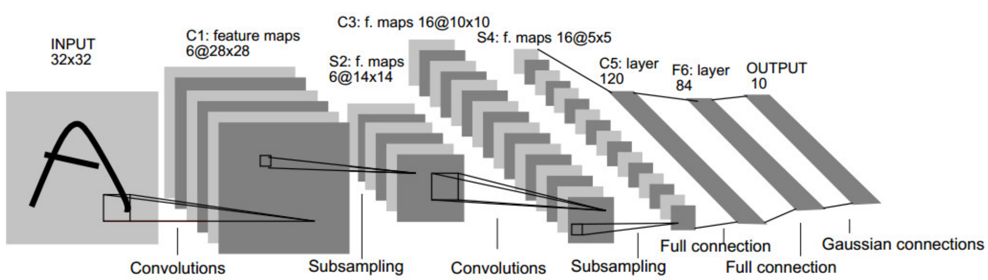

图 5: LeNet-5 结构

LeNet-5 共有 7 层，不包含输入，每层都包含可训练参数；每个层有多个 Feature Map，每个 FeatureMap 通过一种卷积滤波器提取输入的一种特征。

```
with tf.name_scope('conv'):
    conv1 = tf.keras.layers.Conv2D(12, [3,3], strides = 1, padding = 'SAME', name = 'conv1')
    tmpRes = conv1(x_reshaped)
    pool1 = tf.keras.layers.MaxPool2D([3,3], strides = 2, name = 'pool1') # [3,3]
```

```
    tmpRes = pool1(tmpRes)
    conv2 = tf.keras.layers.Conv2D(16, [3,3], strides = 1, padding = 'SAME', name = 'conv2')
    tmpRes = conv2(tmpRes)
    pool2 = tf.keras.layers.MaxPool2D([3,3], strides = 2, name = 'pool2')
    tmpRes = pool2(tmpRes)
    pool2_flatten = tf.reshape(tmpRes, shape = (-1, 6*6*16))

    fc1 = tf.compat.v1.layers.Dense(256, activation = tf.nn.selu, name = 'fc1')
    tmpRes = fc1(pool2_flatten)
    res_fc1 = tmpRes
    fc2 = tf.keras.layers.Dense(100, activation = tf.nn.selu, name = 'fc2')
    tmpRes = fc2(tmpRes)
    res_fc2 = tmpRes
    logits = tf.keras.layers.Dense(10, activation = tf.nn.selu, name = 'output')
    tmpRes = logits(res_fc2)
    res_logits = tmpRes
```

3.2.4 训练并检测手写数字识别结果正确性

接着创建完损失函数后开始训练和测试。

```
# train
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    out = []
    for epoch in range(5):
        for x_batch, y_batch in shuffle_batch(x_train, y_train, batch_size):
            x_batch = np.reshape(x_batch, [-1, 28, 28])
            sess.run(training_op, feed_dict = {x:x_batch, y:y_batch})
        if epoch % 1 == 0:
            batch_acc = accuracy.eval(feed_dict = {x:x_batch, y:y_batch})
            x_test = np.reshape(x_test, [-1, 28, 28])
            val_acc = accuracy.eval(feed_dict = {x:x_test, y:y_test})
            print(epoch, "Batch Accuracy = ",batch_acc," Validation Accuracy = ",val_acc)
            outputs = sess.run(res_logits, feed_dict = {x:x_test})
            out.append(outputs)

# test accuracy

y_int_test = list(map(int,y_test))
y_hat = np.argmax(outputs, axis = 1)
from sklearn.metrics import accuracy_score
acc_score = accuracy_score(y_int_test, y_hat)
print(acc_score)
```

3.2.5 添加 dropout 和批量标准化重新训练

现在为网络添加两个新元素：dropout 和批量标准化，同时开始使用增强后的数据集进行训练。

```
# new with shifted & flipped network
with tf.name_scope('conv'):
    # conv layer
    conv1 = tf.keras.layers.Conv2D(12, [3,3], strides = 1, padding = 'SAME', name = 'conv1')
    tmpRes = conv1(x_reshaped)
    pool1 = tf.keras.layers.MaxPool2D([3,3], strides = 2, name = 'pool1') # [3,3] is ?
    tmpRes = pool1(tmpRes)
    res_pool1 = tmpRes
    # momentum & renorm_momentum
    bn1 = tf.compat.v1.layers.batch_normalization(res_pool1, momentum = 0.9, training =
        bn1_train)
    # tmpRes = bn1(res_pool1, training = bn1_train)
    dropout1 = tf.compat.v1.keras.layers.Dropout(0.5)
    tmpRes = dropout1(bn1, training = drop1)

    conv2 = tf.keras.layers.Conv2D(16, [3,3], strides = 1, padding = 'SAME', name = 'conv2')
    tmpRes = conv2(res_pool1) #Attention! use pool1
    pool2 = tf.keras.layers.MaxPool2D([3,3], strides = 2, name = 'pool2')
    tmpRes = pool2(tmpRes)
    res_pool2 = tmpRes
    #bn2 = tf.keras.layers.BatchNormalization(momentum = 0.9)
    bn2 = tf.compat.v1.layers.batch_normalization(res_pool2, momentum = 0.9, training =
        bn2_train)
    # tmpRes = bn2(tmpRes, training = bn2_train)
    # res_bn2 = tmpRes
    dropout2 = tf.compat.v1.keras.layers.Dropout(0.5)
    tmpRes = dropout2(bn2, training = drop2)
    res_dropout2 = tmpRes

    bn2_flatten = tf.reshape(tmpRes, shape = (-1, 6*6*16)) #res_pool2

    fc1 = tf.compat.v1.layers.Dense(256, activation = tf.nn.selu, name = 'fc1')
    tmpRes = fc1(bn2_flatten)
    res_fc1 = tmpRes
    fc2 = tf.keras.layers.Dense(100, activation = tf.nn.selu, name = 'fc2')
    tmpRes = fc2(res_fc1)
    res_fc2 = tmpRes
    logits = tf.keras.layers.Dense(10, activation = tf.nn.selu, name = 'output')
    tmpRes = logits(res_fc2)
    res_logits = tmpRes

# train
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    out = []
    for epoch in range(20):
        for x_batch, y_batch in shuffle_batch(final_x, final_y, batch_size):
            x_batch = np.reshape(x_batch, [-1, 28, 28])
            sess.run([training_op,extra_update_ops], feed_dict = {x:x_batch, y:y_batch})
```

```
        if epoch % 1 == 0:
            batch_acc = accuracy.eval(feed_dict = {x:x_batch, y:y_batch})
            x_test = np.reshape(x_test, [-1, 28, 28])
            val_acc = accuracy.eval(feed_dict = {bn1_train: False, bn2_train: False,
                                                  drop1: False, drop2: False,
                                                  x:x_test, y:y_test})
            print(epoch, "Batch Accuracy = ",batch_acc," Validation Accuracy = ",val_acc)
            outputs = sess.run(res_logits, feed_dict = {bn1_train: False, bn2_train: False,
                                                         drop1: False, drop2: False,
                                                         x:x_test})
            out.append(outputs)
```

四、 主要仪器设备

计算机，anaconda, Jupyter Notebook

五、 实验结果

1. 编译运行

在 Jupyter Notebook 中依次序运行代码块。

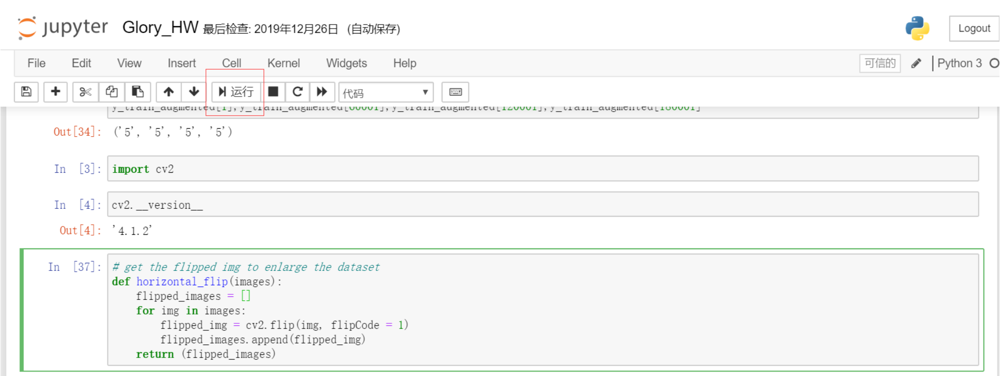

图 6: 运行代码

2. 运行结果

未添加 dropout 和批量标准化，未使用增强数据集，训练 20 次的过程和结果。从图中可以发现，训练到第 20 次的时候得到的正确率已经接近 99%。

```
In [34]: with tf.compat.v1.Session() as sess:
             sess.run(tf.compat.v1.global_variables_initializer())
             out = []
             for epoch in range(20):
                 for x_batch, y_batch in shuffle_batch(x_train, y_train, batch_size):
                     x_batch = np.reshape(x_batch, [-1, 28, 28])
                     sess.run(training_op, feed_dict = {x:x_batch, y:y_batch})
                 if epoch % 1 == 0:
                     batch_acc = accuracy.eval(feed_dict = {x:x_batch, y:y_batch})
                     x_test = np.reshape(x_test, [-1, 28, 28])
                     val_acc = accuracy.eval(feed_dict = {x:x_test, y:y_test})
                     print(epoch, "Batch Accuracy = ",batch_acc," Validation Accuracy = ",val_acc)
                     outputs = sess.run(res_logits, feed_dict = {x:x_test})
                     out.append(outputs)

0 Batch Accuracy =  0.9609375   Validation Accuracy =  0.9568
1 Batch Accuracy =  0.96875   Validation Accuracy =  0.9787
2 Batch Accuracy =  1.0   Validation Accuracy =  0.9796
3 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9859
4 Batch Accuracy =  1.0   Validation Accuracy =  0.9839
5 Batch Accuracy =  1.0   Validation Accuracy =  0.9861
6 Batch Accuracy =  1.0   Validation Accuracy =  0.9853
7 Batch Accuracy =  1.0   Validation Accuracy =  0.9871
8 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9855
9 Batch Accuracy =  1.0   Validation Accuracy =  0.9853
10 Batch Accuracy =  1.0   Validation Accuracy =  0.9841
11 Batch Accuracy =  1.0   Validation Accuracy =  0.9866
12 Batch Accuracy =  1.0   Validation Accuracy =  0.9875
13 Batch Accuracy =  1.0   Validation Accuracy =  0.9875
14 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9808
15 Batch Accuracy =  0.984375   Validation Accuracy =  0.9837
16 Batch Accuracy =  1.0   Validation Accuracy =  0.9874
17 Batch Accuracy =  1.0   Validation Accuracy =  0.9867
18 Batch Accuracy =  1.0   Validation Accuracy =  0.9871
19 Batch Accuracy =  1.0   Validation Accuracy =  0.9886
```

图 7: 训练结果 1

而后我使用 accuracy_score 来评估正确性，可以看到得分达到了 98.86。

```
In [39]: from sklearn.metrics import accuracy_score

In [40]: acc_score = accuracy_score(y_int_test, y_hat)
         print(acc_score)

         0.9886
```

图 8: 正确性 1

添加 dropout 和批量标准化，使用增强数据集，训练 20 次的过程和结果。从图中可以发现，训练到第 20 次的时候得到的正确率已经超过 99%。效果好于未改进的情况。

```
In [356]: with tf.compat.v1.Session() as sess:
              sess.run(tf.compat.v1.global_variables_initializer())
              out = []
              for epoch in range(20):
                  for x_batch, y_batch in shuffle_batch(final_x, final_y, batch_size):
                      x_batch = np.reshape(x_batch, [-1, 28, 28])
                      sess.run([training_op,extra_update_ops], feed_dict = {x:x_batch, y:y_batch})
                  if epoch % 1 == 0:
                      batch_acc = accuracy.eval(feed_dict = {x:x_batch, y:y_batch})
                      x_test = np.reshape(x_test, [-1, 28, 28])
                      val_acc = accuracy.eval(feed_dict = {bn1_train: False, bn2_train: False,
                                                           drop1: False, drop2: False,
                                                           x:x_test, y:y_test})
                      print(epoch, "Batch Accuracy = ",batch_acc," Validation Accuracy = ",val_acc)
                      outputs = sess.run(res_logits, feed_dict = {bn1_train: False, bn2_train: False,
                                                                  drop1: False, drop2: False,
                                                                  x:x_test})
                      out.append(outputs)

0 Batch Accuracy =  1.0   Validation Accuracy =  0.9854
1 Batch Accuracy =  0.984375   Validation Accuracy =  0.9905
2 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9907
3 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9915
4 Batch Accuracy =  0.984375   Validation Accuracy =  0.992
5 Batch Accuracy =  0.9609375   Validation Accuracy =  0.9923
6 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9936
7 Batch Accuracy =  0.9921875   Validation Accuracy =  0.992
8 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9941
9 Batch Accuracy =  1.0   Validation Accuracy =  0.9938
10 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9932
11 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9941
12 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9939
13 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9943
14 Batch Accuracy =  1.0   Validation Accuracy =  0.9935
15 Batch Accuracy =  0.9765625   Validation Accuracy =  0.9931
16 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9932
17 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9939
18 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9942
19 Batch Accuracy =  0.9921875   Validation Accuracy =  0.9928
```

图 9: 训练结果 2

而后我使用 accuracy_score 来评估正确性，可以看到得分达到了 99.28。

```
In [373]: from sklearn.metrics import accuracy_score

In [374]: acc_score = accuracy_score(y_int_test, y_hat)
          print(acc_score)

0.9928
```

图 10: 正确性 2

六、 实验结果分析

通过实验的结果数据我们可以认识到 LeNet-5 确实是一种用于手写体字符识别的非常高效的卷积神经网络，在没有数据增强和其他改变的情况下，仅仅对参数稍加改变以适应输入，在经过 20 次的训练后就可以得到接近 99% 的识别成功率。在添加了 dropout 和批量标准化，更是超过了 99%。

经过本次实验的 CNN 入门实践，我对 CNN 中的各种概念，卷积层、池化层和全连接层有了更加深入的了解。并很好巩固了课堂上学习的知识，比如输入，输出，不同参数代表的意义，feature map 大小的计算等等。总的来说，经过本次实验，我触及到了许多不曾学习过的知识领域，也认识到自己在相关数学知识上的不足，获益匪浅。与此同时，本次实验也激发了我更多地去了解 CNN 的兴趣，希望能在今后的学习生活中对其进行更深入的学习。