1. 实验内容

MNIST手写数字数据集来源于是美国国家标准与技术研究所，是著名的公开数据集之一。MNIST手写数字数据集包含有60000张图片作为训练集数据，由250个不同职业的人纯手写绘制。其中10000张图片作为测试集数据，且每一个训练元素都是28×28像素的手写数字图片，每一张图片代表的是从0到9中的每个数字。在keras中内置了MNIST数据集，我们只需导入相关库并使用代码mnist.load\_data()即可下载MNIST数据集。

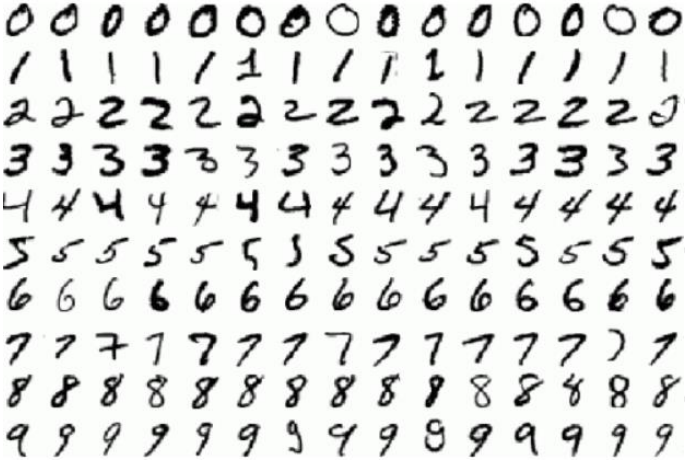


图1 MNIST数据集示例

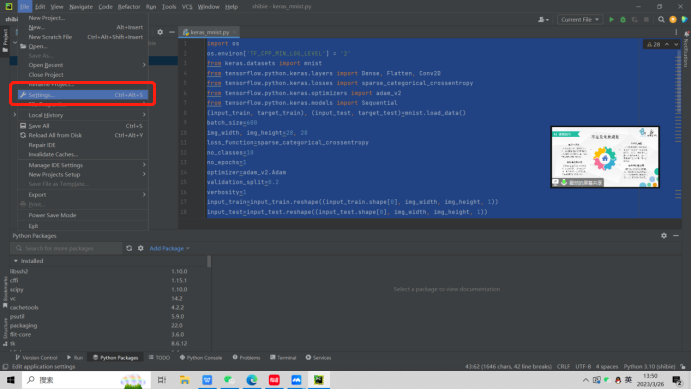
Tensorflow是一个深度学习库，由Google开发，可以对定义在Tensor(张量)上的函数自动求导。Tensor(张量)意味着 N 维数组，Flow(流)意味着基于数据流图的计算，TensorFlow即为张量从图的一端流动到另一端。它的一大亮点是支持异构设备分布式计算，它能够在各个平台上自动运行模型，从电话、单个CPU / GPU到成百上千GPU卡组成的分布式系统。支持CNN、RNN和LSTM算法，是目前在Image，NLP较为流行的深度神经网络模型。

Tensorflow有以下有点，第一，基于Python，写的很快并且具有可读性。第二，在多GPU系统上的运行更为顺畅。第三，代码编译效率较高。第四，社区发展的非常迅速并且活跃。第五，能够生成显示网络拓扑结构和性能的可视化图。

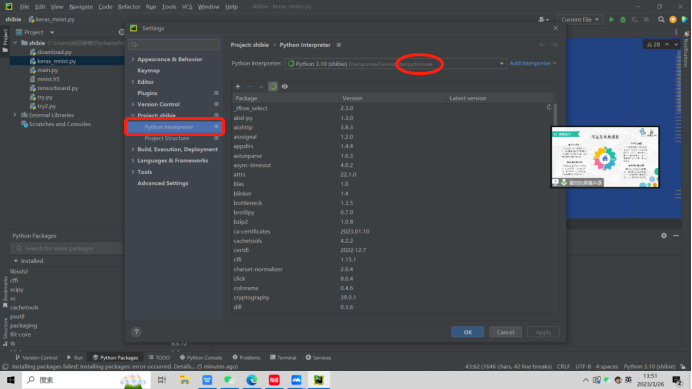
1. 运行结果

环境搭建（这里给的是2.9.0之后tensorflow版本的，2.5.0版本的tensorflow可以直接第一段代码）：

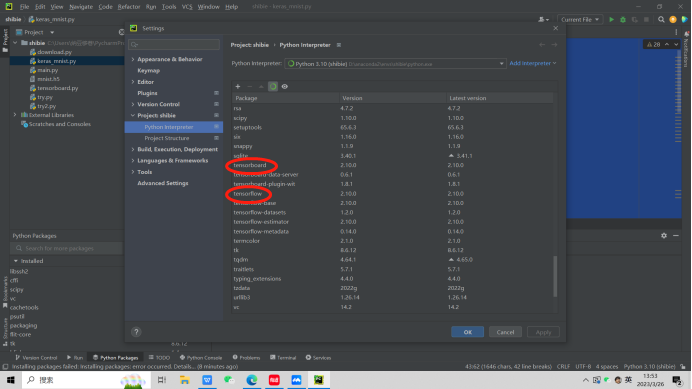
按照ppt先走一遍，然后到pycharm里，如下图所示，找到’settings’这个选项，进入settings界面



在界面里找到python interpreter，然后将自己的conda.exe的路径放入解释器里（注意一定是conda.exe，不是python.exe)

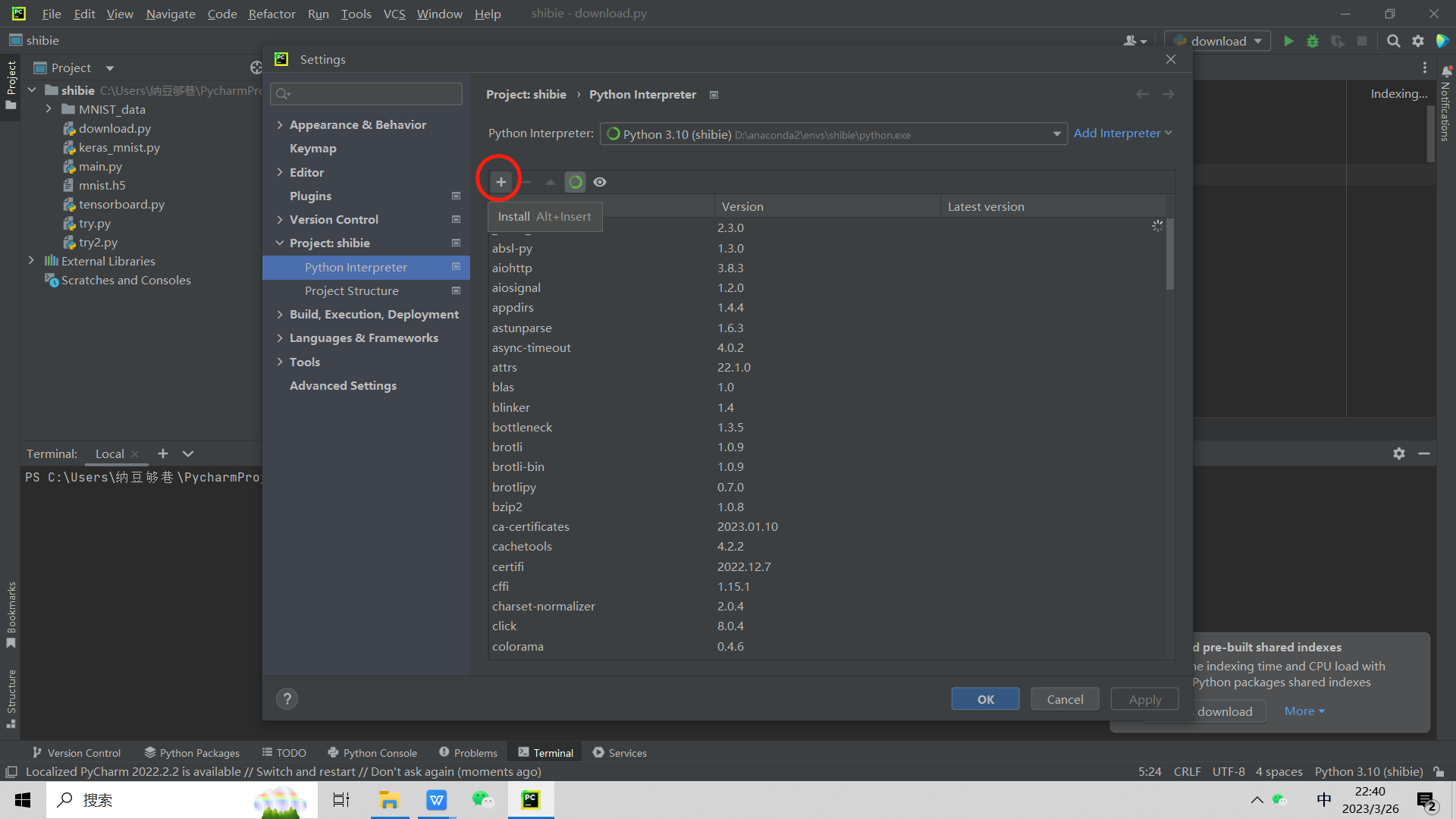


加入解释器后，看一下自己有没有安装成功tensorflow的包（我这张图里还加了tensorflow.datasets，一般都需要加，因为老师给的代码里是有datasets模块再用的）



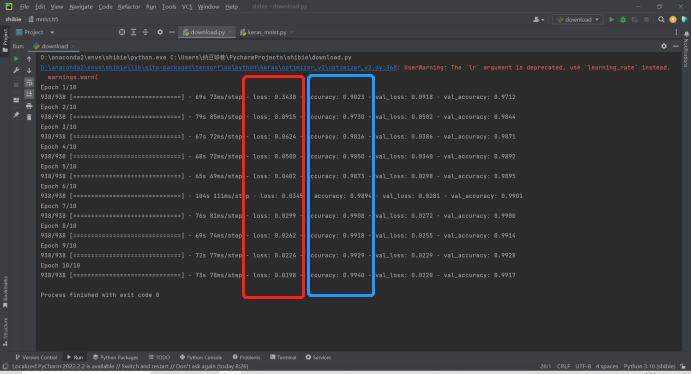
更改代码，主要是之前版本的有些路径有所修改，所以你无法成功import这些包，具体的都可以在博客上搜到，代码详见附录第二段代码，复制过去应该就可以正常运行了。(这个是代码修改后的版本，无法运行时你可以试一下第一段代码，一般都是可以运行的）

如果你甚至发现你的conda环境里面没有你之前下载过的tensorflow（此时你已经照着老师ppt完成了conda环境的配置），可以在pycharm中的settings环境中python interpreter中直接点击加号，搜tensorflow，直接下载，也不影响你后续第二段代码的运行，见下图。



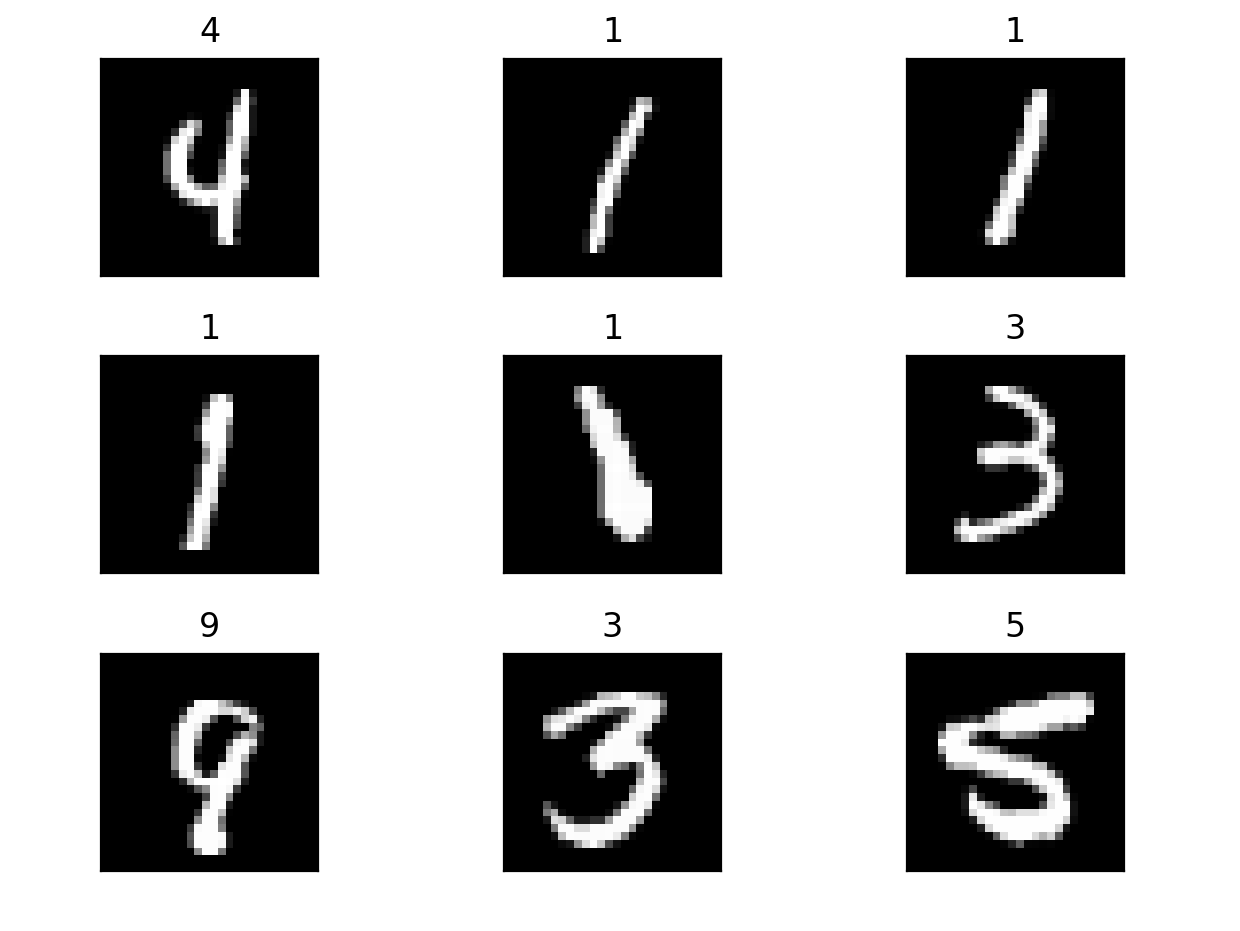
如果以上不能解决你的问题，你还可以手动降低tensorflow的版本（https://blog.csdn.net/qq\_42535133/article/details/105381980），下降到2.5.0就可以正常运行老师的代码了。

实验结果：



运行过程图

结果可视化部分，这里导入了一个现成的工具包对结果进行绘图（matplotlib），自编函数进行结果绘图，可以看到识别的结果还是很准确的（至少在这些样例上），这个matplotlib的包的下载可以参考之前重新下载tensorflow包的那张图，直接在conda环境里下载这个包就可以正常导入运行了。



测试集结果图（代码见代码第三段）

不得不说，tensorflow这个版本更迭很快，但是经常出现新版本无法正常兼容老版本的问题，所以更新的就感觉比较鸡肋，不过问题还是可以解决的。

1. 总结分析

迭代十次之后，可以看到在测试集上第十次迭代时准确率已达99.40%，效果已经比较好，针对测试集利用matplotlib包编写自编函数也得出了相应的结果图，手写识别效果良好。

在实验过程中，不断调试环境和代码，锻炼了个人电脑使用能力。熟练联系了pycharm的使用。

1. 附录（源程序）

**1)第一段示例代码**

# 手写数字识别 -- CNN神经网络训练

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'

import matplotlib as plt

import tensorflow as tf

from tensorflow.python.keras.models import Sequential

from tensorflow.python.keras.layers import Dense, Dropout, Convolution2D, MaxPooling2D, Flatten

from tensorflow.python.keras.optimizers import adam\_v2

# 1、载入数据

mnist = tf.keras.datasets.mnist

(train\_data, train\_target), (test\_data, test\_target) = mnist.load\_data()

# 2、改变数据维度

train\_data = train\_data.reshape(-1, 28, 28, 1)

test\_data = test\_data.reshape(-1, 28, 28, 1)

# 注：在TensorFlow中，在做卷积的时候需要把数据变成4维的格式

# 这4个维度分别是：数据数量，图片高度，图片宽度，图片通道数

# 3、归一化（有助于提升训练速度）

train\_data = train\_data / 255.0

test\_data = test\_data / 255.0

# 4、独热编码

train\_target = tf.keras.utils.to\_categorical(train\_target, num\_classes=10)

test\_target = tf.keras.utils.to\_categorical(test\_target, num\_classes=10) # 10种结果

# 5、搭建CNN卷积神经网络

model = Sequential()

# 5-1、第一层：卷积层+池化层

# 第一个卷积层

model.add(Convolution2D(input\_shape=(28, 28, 1), filters=32, kernel\_size=5, strides=1, padding='same', activation='relu'))

# 卷积层 输入数据 滤波器数量 卷积核大小 步长 填充数据(same padding) 激活函数

# 第一个池化层 # pool\_size

model.add(MaxPooling2D(pool\_size=2, strides=2, padding='same', ))

# 池化层(最大池化) 池化窗口大小 步长 填充方式

# 5-2、第二层：卷积层+池化层

# 第二个卷积层

model.add(Convolution2D(64, 5, strides=1, padding='same', activation='relu'))

# 64:滤波器个数 5:卷积窗口大小

# 第二个池化层

model.add(MaxPooling2D(2, 2, 'same'))

# 5-3、扁平化 （相当于把(64,7,7,64)数据->(64,7\*7\*64)）

model.add(Flatten())

# 5-4、第三层：第一个全连接层

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

# 5-5、第四层：第二个全连接层（输出层）

model.add(Dense(10, activation='softmax'))

# 6、编译

model.compile(optimizer=adam\_v2.Adam(lr=1e-4), loss='categorical\_crossentropy', metrics=['accuracy'])

# 优化器(adam) 损失函数(交叉熵损失函数) 标签

# 7、训练

model.fit(train\_data, train\_target, batch\_size=64, epochs=10, validation\_data=(test\_data, test\_target))

# 8、保存模型

model.summary()

model.save('mnist.h5')

**2)第二段段示例代码（老师ppt上给的，经测试，ppt上的代码在2.5.0的tensoflow下可以顺利运行，但是在2.9.0-2.12.0的版本上无法正常运行，主要问题是tensorflow更新，代码内容和函数命名有所更新，用下面这段代码就可以正常运行了）**

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'

from keras.datasets import mnist

from tensorflow.python.keras.layers import Dense, Flatten, Conv2D

from tensorflow.python.keras.losses import sparse\_categorical\_crossentropy

from tensorflow.python.keras.optimizers import adam\_v2

from tensorflow.python.keras.models import Sequential

(input\_train, target\_train), (input\_test, target\_test)=mnist.load\_data()

batch\_size=600

img\_width, img\_height=28, 28

loss\_function=sparse\_categorical\_crossentropy

no\_classes=10

no\_epochs=1

optimizer=adam\_v2.Adam

validation\_split=0.2

verbosity=1

input\_train=input\_train.reshape((input\_train.shape[0], img\_width, img\_height, 1))

input\_test=input\_test.reshape((input\_test.shape[0], img\_width, img\_height, 1))

input\_shape=(img\_width, img\_height, 1)

input\_train=input\_train.astype('float32')

input\_test=input\_test.astype('float32')

input\_train=input\_train/255

input\_test=input\_test/255

model=Sequential()

model.add=(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape))

model.add=(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add=(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dense(no\_classes, activation='softmax'))

model.compile(

loss=loss\_function,

optimizer=optimizer,

metrics=(['accuracy'])

)

model.fit(input\_train, target\_train,

batch\_size=batch\_size,

epochs=no\_epochs,

verbose=verbosity,

validation\_split=validation\_split

)

scores=model.evaluate(input\_train, target\_train, verbose=0)

print("%s: %.2f%%" % (model.metrics\_names[1], scores[1]\*100))

1. **画图代码**

# img: The raw image[pic\_n, channel, size\_x, size\_y]

# label[] label of each img

# fig\_shape[x,y]: the number of images to show is x\*y

from matplotlib import pyplot as plt

def plot\_image(img, label, fig\_shape):

fig = plt.figure()

x\_n = fig\_shape[0]

y\_n = fig\_shape[1]

for i in range(x\_n\*y\_n):

plt.subplot(x\_n, y\_n, i + 1)

plt.tight\_layout()

plt.imshow(img[i][0] \* 0.3081 + 0.1307, cmap='gray', interpolation='none')

plt.title("{}".format(label[i].item()))

plt.xticks([])

plt.yticks([])

plt.show()