本次任务的目的就一个：学习模仿。LeNet是DL领域比较早期的经典模型，值得认真学习模仿。

具体而言，本次TASK需要你：

（1）学习模仿这个经典网络，并对每行代码做注释解释；

（2）模仿这个经典模型的基本思想，创造一个属于你自己的CNN模型，并用于前面的手写数字识别案例，看看最好能做多好，是否有所改进？

（3）请对代码逐行注释，尤其要对model.summary中呈现的参数个数做详细解释。这对帮助大家更好地理解模型结构帮助巨大！

经典CNN网络：LeNet5 介绍

LeNet5介绍

> 所有研究CNN的都必然知道LeNet-5模型，这是第一个正式的卷积神经网络模型，在1998年，作者还是LeCun，文章《Gradient-based learning applied to document recognition》，截止2019年3月6号，google引用量17000+次：

详解LeNet5网络结构

> input: 32*32的灰度图
>
> 第1步：与6个高为5、宽为5、深度为1的卷积核valid卷积，将卷积结果的每一深度加偏置经激活函数处理，得到结果高28，宽28，深度6.
>
> 第2步：进行2*2 的步长为2的valid最大值池化，池化结果，高14，宽14，深度为6
>
> 第3步：先与16个高为5、宽为5、深度为6的卷积核valid卷积，将卷积结果的每一深度加偏置经激活函数处理，得到结果高10、宽10、深度为16
>
> 第4步：进行2*2 的步长为2的valid最大值池化，池化结果，高5，宽5，深度为16
>
> 第5步：将第4步结果拉长成一维向量，其长度为5516=400，然后将这个向量经过一个全连接神经网络处理，该全连接神经网络共有2个隐含层，其中输入层400个神经元，第1个隐含层有120个神经元，第2个隐含层有84个神经元，输出层有10个神经元（因为是10个分类）

LeNet5: Why?

> 卷积核大小：5×5？可不可以是3×3？7×7？100×100？
>
> 卷积核的个数：16？可不可以是32？可不可以是8？
>
> 池化规格：2×2？可不可以是3×3？可不可以是4×4？

Same vs. Valid？他们之间有啥区别？

隐藏层？应该要多少层？中间要多少神经元？

加载MNIST数据并展示

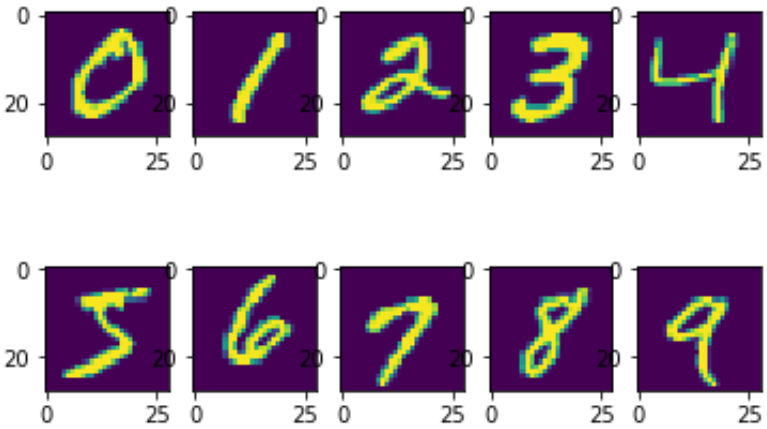
In [1]:

```
from keras.datasets import mnist
(X0,Y0),(X1,Y1) = mnist.load_data(path="/clubear/datasets/mnist.npz")   # 加载MNIST数据
print(X0.shape)     # 将训练集转化为矩阵
from matplotlib import pyplot as plt
plt.figure()                # 绘制画布
fig,ax = plt.subplots(2,5)
ax=ax.flatten()
for i in range(10):
    Im=X0[Y0==i][0]
    ax[i].imshow(Im)
plt.show()              # 加载10个数据图像
```

Using TensorFlow backend.

(60000, 28, 28)

<Figure size 432x288 with 0 Axes>



数据处理与准备工作

In [2]:

```
from keras.utils import np_utils
N0=X0.shape[0];N1=X1.shape[0]     # 转化成矩阵
print([N0,N1])
X0 = X0.reshape(N0,28,28,1)/255
X1 = X1.reshape(N1,28,28,1)/255     # 返回一个和元素相同的n维数组
YY0 = np_utils.to_categorical(Y0)
YY1 = np_utils.to_categorical(Y1)    # 将整型的类别标签转为onehot编码
YY1
```

[60000, 10000]

Out[2]:

```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
```

```
[0., 1., 0., ..., 0., 0., 0.],
...,
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

LeNet5：代码实现

In [3]:

```
from keras.layers import Conv2D,Dense,Flatten,Input,MaxPooling2D
from keras import Model

input_layer = Input([28,28,1])        # 输入矩阵维度
x = input_layer
x = Conv2D(6,[5,5],padding = "same", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)    # 设定池化层为2*2取最大值
x = Conv2D(16,[5,5],padding = "valid", activation = 'relu')(x)
x = MaxPooling2D(pool_size = [2,2], strides = [2,2])(x)
x = Flatten()(x)        # 将数据展平为普通的一维格式
x = Dense(120,activation = 'relu')(x)   # 设定一个普通的全连接层
x = Dense(84,activation = 'relu')(x)
x = Dense(10,activation = 'softmax')(x)   # 设定输出层
output_layer=x
model=Model(input_layer,output_layer)     # 构建模型
model.summary()              # #输出模型各层的参数状况
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

输入层为长度为28x28x1的矩阵，消耗了400个参数，乘以120，再加上截距项，一共消耗了61,706个参数.

LeNet5：编译运行

In [4]:

```
model.compile(loss = 'categorical_crossentropy',optimizer='adam',metrics = ['accuracy']) ##损失函数；优化器；准确率
model.fit(X0,YY0,epochs = 10,batch_size = 200,validation_data=[X1,YY1])     # 输入数据和标签,输出损失和精确度.
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 3s 57us/step - loss: 0.3598 - accuracy: 0.8984 - val_los
s: 0.1088 - val_accuracy: 0.9691
Epoch 2/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0962 - accuracy: 0.9706 - val_los
s: 0.0679 - val_accuracy: 0.9778
Epoch 3/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0665 - accuracy: 0.9795 - val_los
s: 0.0585 - val_accuracy: 0.9827
Epoch 4/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0529 - accuracy: 0.9833 - val_los
s: 0.0492 - val_accuracy: 0.9853
Epoch 5/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0445 - accuracy: 0.9859 - val_los
s: 0.0463 - val_accuracy: 0.9860
Epoch 6/10
60000/60000 [==============================] - 2s 38us/step - loss: 0.0370 - accuracy: 0.9885 - val_los
s: 0.0533 - val_accuracy: 0.9828
Epoch 7/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0312 - accuracy: 0.9897 - val_los
s: 0.0372 - val_accuracy: 0.9889
Epoch 8/10
60000/60000 [==============================] - 2s 38us/step - loss: 0.0268 - accuracy: 0.9911 - val_los
s: 0.0361 - val_accuracy: 0.9878
Epoch 9/10
60000/60000 [==============================] - 2s 39us/step - loss: 0.0236 - accuracy: 0.9922 - val_los
s: 0.0332 - val_accuracy: 0.9891
Epoch 10/10
60000/60000 [==============================] - 2s 38us/step - loss: 0.0221 - accuracy: 0.9927 - val_los
s: 0.0382 - val_accuracy: 0.9888
```

Out[4]:

```
<keras.callbacks.callbacks.History at 0x7fd904beb7d0>
```

思考问题：LeNet可以如何修改？

In [2]:

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.utils import np_utils
from keras.layers import Dense, Dropout, Convolution2D, MaxPooling2D, Flatten
from keras.optimizers import Adam
# 载入数据
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [3]:

```
x_train = x_train.reshape(-1, 28, 28, 1)/255.0
x_test = x_test.reshape(-1, 28, 28, 1)/255.0
```

In [4]:

```
# 换one hot格式
y_train = np_utils.to_categorical(y_train, num_classes=10)
y_test = np_utils.to_categorical(y_test, num_classes=10)
```

In [5]:

```
# 定义顺序模型
model = Sequential()
# 第一个卷积层
model.add(Convolution2D(
    input_shape = (28, 28, 1),     # 输入平面
    filters = 32,                  # 卷积核/滤波器个数
    kernel_size = 5,               # 卷积窗口大小
    strides = 1,                   # 步长
    padding = "same",              # 方式
    activation = "relu"))          # 激活函数
```

In [6]:

```
# 第一个池化层
model.add(MaxPooling2D(
    pool_size = 2,
    strides = 2,
    padding = 'same',
))
```

In [8]:

```
# 第二个卷积层
model.add(Convolution2D(64,5,strides=1,padding='same',activation = 'relu'))
# 第二个池化层
model.add(MaxPooling2D(2,2,'same'))
# 把第二个池化层的输出扁平化为1维
model.add(Flatten())
```

In [9]:

```
# 第一个全连接层
model.add(Dense(1024,activation = 'relu'))
# Dropout
model.add(Dropout(0.5))
# 第二个全连接层
model.add(Dense(10,activation='softmax'))
```

In [10]:

```
# 定义优化器
adam = Adam(lr=1e-4)
# 定义优化器，loss function，训练过程中计算准确率
model.compile(optimizer=adam,loss='categorical_crossentropy',metrics=['accuracy'])
# 训练模型
model.fit(x_train,y_train,batch_size=64,epochs=10)
```

```
Epoch 1/10
938/938 [==============================] - 134s 143ms/step - loss: 0.3818 - accuracy: 0.8832 - loss: 0
Epoch 2/10
938/938 [==============================] - 129s 137ms/step - loss: 0.0958 - accuracy: 0.9703
Epoch 3/10
938/938 [==============================] - 127s 136ms/step - loss: 0.0683 - accuracy: 0.9792
```

Epoch 4/10
938/938 [==============================] - 135s 144ms/step - loss: 0.0536 - accuracy: 0.9840
Epoch 5/10
938/938 [==============================] - 135s 144ms/step - loss: 0.0444 - accuracy: 0.9863
Epoch 6/10
938/938 [==============================] - 138s 147ms/step - loss: 0.0370 - accuracy: 0.9884
Epoch 7/10
938/938 [==============================] - 143s 152ms/step - loss: 0.0303 - accuracy: 0.9909
Epoch 8/10
938/938 [==============================] - 143s 152ms/step - loss: 0.0281 - accuracy: 0.9912
Epoch 9/10
938/938 [==============================] - 129s 138ms/step - loss: 0.0236 - accuracy: 0.9926
Epoch 10/10
938/938 [==============================] - 140s 149ms/step - loss: 0.0210 - accuracy: 0.9934

Out[10]:

<tensorflow.python.keras.callbacks.History at 0x1ca3fb46a88>

In [11]:

```
# 评估模型
loss,accuracy = model.evaluate(x_test,y_test)
print('test loss',loss)
print('test accuracy',accuracy)
```

313/313 [==============================] - 3s 10ms/step - loss: 0.0228 - accuracy: 0.9922
test loss 0.022842055186629295
test accuracy 0.9922000169754028