CNN Train Demo

Jin Xia (2020-8-5)

In [1]:
```
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D,AveragePooling2D
from keras.models import model_from_json
import keras.backend as K
from keras.callbacks import LearningRateScheduler
from keras.callbacks import ReduceLROnPlateau
import pickle
import sys,os
import time
import pandas as pd
from mlxtend.data import loadlocal_mnist
import joon.os
from keras import utils as np_utils
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

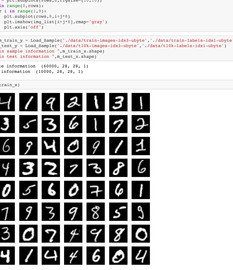
In [2]:
```
def load_Sample(img_file,label_file):
    raw_x,raw_y = loadlocal_mnist(images_path=img_file, labels_path=label_file)
    x = []
    y = []
    for label,item in zip(raw_y,raw_x):
        y.append(label)
        x.append(item)
    x = np.array(x)
    n_classes = len(np.unique(y))
    y = np_utils.to_categorical(y, n_classes)
    x = x.reshape(x.shape[0], 28, 28, 1)
    return x,y
```

In [3]:
```
def ShowImg(img_list):
    rows = 2
    if len(img_list) < 6:
        rows = int(len(img_list)/3)
    fig,ax = plt.subplots(rows,3,figsize=(10,10))
    for i in range(0,rows):
        for j in range(0,3):
            plt.subplot(rows,3,i*3+j)
            plt.imshow(img_list[i*3+j],cmap='gray')
            plt.axis('off')
```

In [4]:
```
m_train_x,m_train_y = load_Sample('./data/train-images-idx3-ubyte','./data/train-labels-idx1-ubyte')
m_test_x,m_test_y = load_Sample('./data/t10k-images-idx3-ubyte','./data/t10k-labels-idx1-ubyte')
print('Train sample information',m_train_x.shape)
print('Train test information',m_test_x.shape)
```

```
Train sample information (60000, 28, 28, 1)
Train test information (10000, 28, 28, 1)
```

In [5]:
```
ShowImg(m_train_x)
```



In [6]:
```
m_model = Sequential()
m_model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(28,28,1)))
m_model.add(Conv2D(32, (5, 5), activation='relu'))
m_model.add(MaxPooling2D(pool_size=(2,2)))
m_model.add(Dropout(0.25))
m_model.add(Flatten()
m_model.add(Dense(10, activation='softmax'))
m_model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

In [7]:
```
measure = m_model.fit(m_train_x,m_train_y,batch_size=100,epochs=50,verbose=1,validation_data=(m_test_x,m_test_y))
```

```
Epoch 1/50
600/600 [==============================] - 21s 35ms/step - loss: 0.5988 - accuracy: 0.9109 - val_loss: 0.0871 - val_a
ccuracy: 0.9737
Epoch 2/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0960 - accuracy: 0.9711 - val_loss: 0.0587 - val_a
ccuracy: 0.9805
Epoch 3/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0737 - accuracy: 0.9768 - val_loss: 0.0566 - val_a
ccuracy: 0.9819
Epoch 4/50
600/600 [==============================] - 22s 37ms/step - loss: 0.0612 - accuracy: 0.9808 - val_loss: 0.0540 - val_a
ccuracy: 0.9838
Epoch 5/50
600/600 [==============================] - 23s 38ms/step - loss: 0.0526 - accuracy: 0.9830 - val_loss: 0.0583 - val_a
ccuracy: 0.9825
Epoch 6/50
600/600 [==============================] - 22s 37ms/step - loss: 0.0470 - accuracy: 0.9848 - val_loss: 0.0460 - val_a
ccuracy: 0.9858
Epoch 7/50
600/600 [==============================] - 21s 36ms/step - loss: 0.0434 - accuracy: 0.9868 - val_loss: 0.0593 - val_a
ccuracy: 0.9820
Epoch 8/50
600/600 [==============================] - 21s 36ms/step - loss: 0.0425 - accuracy: 0.9864 - val_loss: 0.0511 - val_a
ccuracy: 0.9854
Epoch 9/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0384 - accuracy: 0.9874 - val_loss: 0.0540 - val_a
ccuracy: 0.9856
Epoch 10/50
600/600 [==============================] - 21s 36ms/step - loss: 0.0376 - accuracy: 0.9878 - val_loss: 0.0593 - val_a
ccuracy: 0.9841
Epoch 11/50
600/600 [==============================] - 21s 36ms/step - loss: 0.0320 - accuracy: 0.9893 - val_loss: 0.0617 - val_a
ccuracy: 0.9842
Epoch 12/50
600/600 [==============================] - 21s 36ms/step - loss: 0.0339 - accuracy: 0.9890 - val_loss: 0.0598 - val_a
ccuracy: 0.9853
Epoch 13/50
600/600 [==============================] - 21s 35ms/step - loss: 0.0315 - accuracy: 0.9900 - val_loss: 0.0573 - val_a
ccuracy: 0.9857
Epoch 14/50
600/600 [==============================] - 22s 37ms/step - loss: 0.0287 - accuracy: 0.9910 - val_loss: 0.0713 - val_a
ccuracy: 0.9837
Epoch 15/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0310 - accuracy: 0.9901 - val_loss: 0.0612 - val_a
ccuracy: 0.9865
Epoch 16/50
600/600 [==============================] - 21s 35ms/step - loss: 0.0245 - accuracy: 0.9922 - val_loss: 0.0716 - val_a
ccuracy: 0.9862
Epoch 17/50
600/600 [==============================] - 21s 36ms/step - loss: 0.0279 - accuracy: 0.9912 - val_loss: 0.0656 - val_a
ccuracy: 0.9862
Epoch 18/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0239 - accuracy: 0.9927 - val_loss: 0.0674 - val_a
ccuracy: 0.9872
Epoch 19/50
600/600 [==============================] - 21s 35ms/step - loss: 0.0244 - accuracy: 0.9920 - val_loss: 0.0615 - val_a
ccuracy: 0.9860
Epoch 20/50
600/600 [==============================] - 21s 35ms/step - loss: 0.0216 - accuracy: 0.9932 - val_loss: 0.0648 - val_a
ccuracy: 0.9871
Epoch 21/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0241 - accuracy: 0.9924 - val_loss: 0.0722 - val_a
ccuracy: 0.9873
Epoch 22/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0216 - accuracy: 0.9930 - val_loss: 0.0736 - val_a
ccuracy: 0.9871
Epoch 23/50
600/600 [==============================] - 22s 37ms/step - loss: 0.0233 - accuracy: 0.9933 - val_loss: 0.0659 - val_a
ccuracy: 0.9875
Epoch 24/50
600/600 [==============================] - 22s 36ms/step - loss: 0.0221 - accuracy: 0.9933 - val_loss: 0.0788 - val_a
ccuracy: 0.9859
Epoch 25/50
600/600 [==============================] - 24s 41ms/step - loss: 0.0215 - accuracy: 0.9934 - val_loss: 0.0648 - val_a
ccuracy: 0.9873
Epoch 26/50
600/600 [==============================] - 30s 50ms/step - loss: 0.0238 - accuracy: 0.9931 - val_loss: 0.0731 - val_a
ccuracy: 0.9868
Epoch 27/50
600/600 [==============================] - 32s 54ms/step - loss: 0.0202 - accuracy: 0.9944 - val_loss: 0.0717 - val_a
ccuracy: 0.9875
Epoch 28/50
600/600 [==============================] - 32s 54ms/step - loss: 0.0193 - accuracy: 0.9944 - val_loss: 0.0751 - val_a
ccuracy: 0.9859
Epoch 29/50
600/600 [==============================] - 30s 50ms/step - loss: 0.0211 - accuracy: 0.9936 - val_loss: 0.0770 - val_a
ccuracy: 0.9870
Epoch 30/50
600/600 [==============================] - 30s 51ms/step - loss: 0.0185 - accuracy: 0.9944 - val_loss: 0.0795 - val_a
ccuracy: 0.9876
Epoch 31/50
600/600 [==============================] - 30s 51ms/step - loss: 0.0200 - accuracy: 0.9943 - val_loss: 0.0825 - val_a
ccuracy: 0.9866
Epoch 32/50
600/600 [==============================] - 30s 51ms/step - loss: 0.0174 - accuracy: 0.9951 - val_loss: 0.0778 - val_a
ccuracy: 0.9882
Epoch 33/50
600/600 [==============================] - 30s 50ms/step - loss: 0.0180 - accuracy: 0.9949 - val_loss: 0.0789 - val_a
ccuracy: 0.9875
Epoch 34/50
600/600 [==============================] - 30s 51ms/step - loss: 0.0195 - accuracy: 0.9944 - val_loss: 0.0896 - val_a
ccuracy: 0.9861
Epoch 35/50
600/600 [==============================] - 31s 51ms/step - loss: 0.0172 - accuracy: 0.9950 - val_loss: 0.0855 - val_a
ccuracy: 0.9876
Epoch 36/50
600/600 [==============================] - 33s 55ms/step - loss: 0.0200 - accuracy: 0.9944 - val_loss: 0.0861 - val_a
ccuracy: 0.9863
Epoch 37/50
600/600 [==============================] - 33s 55ms/step - loss: 0.0187 - accuracy: 0.9949 - val_loss: 0.0792 - val_a
ccuracy: 0.9882
Epoch 38/50
600/600 [==============================] - 33s 54ms/step - loss: 0.0169 - accuracy: 0.9954 - val_loss: 0.0890 - val_a
ccuracy: 0.9879
Epoch 39/50
600/600 [==============================] - 31s 52ms/step - loss: 0.0178 - accuracy: 0.9952 - val_loss: 0.0930 - val_a
ccuracy: 0.9878
Epoch 40/50
600/600 [==============================] - 31s 52ms/step - loss: 0.0155 - accuracy: 0.9953 - val_loss: 0.0925 - val_a
ccuracy: 0.9882
Epoch 41/50
600/600 [==============================] - 31s 52ms/step - loss: 0.0179 - accuracy: 0.9950 - val_loss: 0.0849 - val_a
ccuracy: 0.9872
Epoch 42/50
600/600 [==============================] - 31s 51ms/step - loss: 0.0187 - accuracy: 0.9950 - val_loss: 0.0932 - val_a
ccuracy: 0.9867
Epoch 43/50
600/600 [==============================] - 28s 47ms/step - loss: 0.0158 - accuracy: 0.9953 - val_loss: 0.0857 - val_a
ccuracy: 0.9878
Epoch 44/50
600/600 [==============================] - 21s 35ms/step - loss: 0.0143 - accuracy: 0.9960 - val_loss: 0.1010 - val_a
ccuracy: 0.9878
Epoch 45/50
600/600 [==============================] - 21s 35ms/step - loss: 0.0189 - accuracy: 0.9951 - val_loss: 0.0976 - val_a
ccuracy: 0.9878
Epoch 46/50
600/600 [==============================] - 20s 34ms/step - loss: 0.0151 - accuracy: 0.9962 - val_loss: 0.0851 - val_a
ccuracy: 0.9888
Epoch 47/50
600/600 [==============================] - 20s 34ms/step - loss: 0.0150 - accuracy: 0.9958 - val_loss: 0.0947 - val_a
ccuracy: 0.9879
Epoch 48/50
600/600 [==============================] - 20s 34ms/step - loss: 0.0177 - accuracy: 0.9957 - val_loss: 0.0932 - val_a
ccuracy: 0.9873
Epoch 49/50
600/600 [==============================] - 20s 34ms/step - loss: 0.0163 - accuracy: 0.9955 - val_loss: 0.0986 - val_a
ccuracy: 0.9884
Epoch 50/50
600/600 [==============================] - 20s 33ms/step - loss: 0.0145 - accuracy: 0.9963 - val_loss: 0.0979 - val_a
ccuracy: 0.9887
```

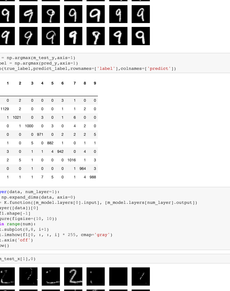
In [8]:
```
epochs = len(measure.history['loss'])
# plt.figure(figsize=(8,4))
# plt.plot(range(0,epochs),y, measure.history['loss'], label='train_loss',color='blue')
# plt.plot(range(0,epochs),y, measure.history['val_loss'], label='test_loss',color='green')
plt.plot(range(0,epochs),y, measure.history['accuracy'], label='train_accuracy',color='red')
plt.plot(range(0,epochs),y, measure.history['val_accuracy'], label='test_accuracy',color='green')
plt.ylabel('accuracy');
plt.xlabel('loss/accuracy');
plt.legend();
plt.show;
```

In [9]:
```
score = m_model.evaluate(m_test_x, m_test_y, verbose=1)
print(score)
```

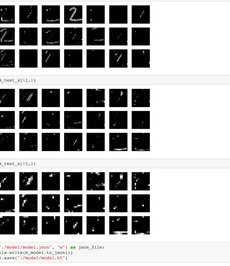
```
313/313 [==============================] - 1s 2ms/step - loss: 0.0979 - accuracy: 0.9887
[0.097867429256432, 0.9886999726295471]
```

In [10]:
```
pred_y = m_model.predict(m_test_x)
pred_score = []
result = []
for i in range(0,len(m_test_x.tolist())):
    img = m_test_x[index]
    pred_label = np.argmax(pred_y[index])
    pred_score = pred_y[index][pred_label]
    if m_test_y[index].tolist() == 1:
        result[pred_label] = 1]
    result[pred_label].append(img)
```

In [11]:
```
ShowImg(m_test_x)
```



In [12]:
```
ShowImg(result[9])
```



In [13]:
```
true_label = np.argmax(m_test_y,axis=1)
predict_label = np.argmax(pred_y,axis=1)
pd.crosstab(true_label,predict_label,rownames=['label'],colnames=['predict'])
```

Out[13]:

predict	0	1	2	3	4	5	6	7	8	9
label										
0	974	0	2	0	0	0	3	1	0	0
1	0	1135	2	0	0	0	1	1	0	0
2	0	1	1021	0	3	0	1	6	0	0
3	0	0	1	1000	0	3	0	4	2	0
4	0	0	0	0	971	0	2	2	2	5
5	1	1	0	5	0	882	1	0	1	1
6	3	3	1	1	7	4	932	0	1	0
7	0	2	5	1	0	0	0	1016	1	3
8	5	0	1	1	0	0	0	1	964	2
9	1	1	1	1	5	0	0	1	4	986

In [14]:
```
def ShowLayer(data, num_layer=1):
    data = np.expand_dims(data, axis=0)
    layer = m_model.layers[num_layer].input, [m_model.layers[num_layer].output]
    f1 = layer(data)[0]
    num = f1.shape[-1]
    plt.figure(figsize=(10, 10))
    for i in range(num):
        plt.subplot(1, num, i+1)
        plt.imshow(f1[0, :, :, i] * 255, cmap='gray')
        plt.axis('off')
    plt.show()
```

In [15]:
```
ShowLayer(m_test_x[1],2)
```


In [16]:
```
ShowLayer(m_test_x[1],1)
```


In [17]:
```
ShowLayer(m_test_x[1],2)
```


In [18]:
```
with open("./model/model.json", "w") as json_file:
    json_file.write(m_model.to_json())
    m_model.save("./model/model.h5")
```