M2608.001300 Machine Learning
Assignment #5 Final Projects (Pytorch)

For understanding of this work, please carefully look at given PPT file.

Note: certain details are missing or ambiguous on purpose, in order to test your knowledge on the related materials. However, if you really feel that something essential is missing and cannot proceed to the next step, then contact the teaching staff with clear description of your problem.

In [1]:

```python
# from google.colab import drive
# drive.mount('/content/drive')

import os
import random

import matplotlib.pyplot as plt
import numpy as np

import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from torch.utils.data import DataLoader,Dataset
from torch.autograd import Variable
from PIL import Image
import resnet
```

Load datasets

In [2]:

```
NUMBER = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
ALPHABET = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
NONE = ['NONE'] # label for empty space
ALL_CHAR_SET = NUMBER + ALPHABET + NONE
ALL_CHAR_SET_LEN = len(ALL_CHAR_SET)
MAX_CAPTCHA = 7

print(ALL_CHAR_SET.index('NONE'))

def encode(a):
    onehot = [0]*ALL_CHAR_SET_LEN
    idx = ALL_CHAR_SET.index(a)
    onehot[idx] += 1
    return onehot

# modified dataset class
class Mydataset(Dataset):
    def __init__(self, img_path, label_path, is_train=True, transform=No
ne):
        self.path = img_path
        self.label_path = label_path
        if is_train:
            self.img = os.listdir(self.path)[:1000]
            self.labels = open(self.label_path, 'r').read().split('\n
')[:-1][:1000]
        else:
            self.img = os.listdir(self.path)[:1000]
            self.labels = open(self.label_path, 'r').read().split('\n
')[:-1][:1000]

        self.transform = transform
        self.max_length = MAX_CAPTCHA

    def __getitem__(self, idx):
        img_path = self.img[idx]
        img = Image.open(f'{self.path}/{self.img[idx]}')
        img = img.convert('L')
        label = self.labels[idx]
        label_oh = []
        # one-hot for each character
        for i in range(self.max_length):
            if i < len(label):
                label_oh += encode(label[i])
            else:
                #label_oh += [0]*ALL_CHAR_SET_LEN
                label_oh += encode('NONE')

        if self.transform is not None:
            img = self.transform(img)
        return img, np.array(label_oh), label

    def __len__(self):
        return len(self.img)

transform = transforms.Compose([
    transforms.Resize([160, 60]),
    transforms.ToTensor(),
########################################################################
######
#                           IMPLEMENT YOUR CODE
#
########################################################################
######
# transforms.Normalize((0.1307, ), (0.3081, ))

########################################################################
######
```

```
36
```

In [3]:

```python
"""Loading DATA"""
# Change to your own data folder path!
# gPath = '/content/drive/My Drive/Colab Notebooks/'
gPath = './'

train_ds = Mydataset(gPath+'Data/train/', gPath+'Data/train.txt',transform=transform)
test_ds = Mydataset(gPath+'Data/test/', gPath+'Data/test.txt', False, transform)
train_dl = DataLoader(train_ds, batch_size=128, num_workers=4)
test_dl = DataLoader(test_ds, batch_size=1, num_workers=4)
```

In [4]:

```python
"""To CUDA for local run"""
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# device = torch.device("cpu")
print(device)


#GPUID = '4' # define GPUID
#os.environ["CUDA_VISIBLE_DEVICES"] = str(GPUID)
```

```
cuda:0
```

Problem 1: Design LSTM model for catcha image recognition. (10 points)

[Captioning Images with CNN and RNN, using PyTorch (https://medium.com/@stepanulyanin/captioning-images-with-pytorch-bc592e5fd1a3)](https://medium.com/@stepanulyanin/captioning-images-with-pytorch-bc592e5fd1a3)

In [5]:

```
class LSTM(nn.Module):
    def __init__(self, cnn_dim, hidden_size, vocab_size, num_layers=1):
        super(LSTM, self).__init__()

        # define the properties
        self.cnn_dim = cnn_dim
        self.hidden_size = hidden_size
        self.vocab_size = vocab_size

        # lstm cell
        self.lstm_cell = nn.LSTMCell(input_size=self.vocab_size, hidden_
size=hidden_size)

        # output fully connected layer
        self.fc_in = nn.Linear(in_features=self.cnn_dim, out_features=se
lf.vocab_size)
        self.fc_out = nn.Linear(in_features=self.hidden_size, out_featur
es=self.vocab_size)

        # embedding layer
        self.embed = nn.Embedding(num_embeddings=self.vocab_size, embedd
ing_dim=self.vocab_size)

        # activations
        self.softmax = nn.Softmax(dim=1)

    def forward(self, features, captions):

        batch_size = features.size(0)
        cnn_dim = features.size(1)

        hidden_state = torch.zeros((batch_size, self.hidden_size)).cuda
()
        cell_state = torch.zeros((batch_size, self.hidden_size)).cuda()

        # define the output tensor placeholder
        outputs = torch.empty((batch_size, captions.size(1), self.vocab_
size)).cuda()

        # embed the captions
        captions_embed = self.embed(captions)

########################################################################
######
#                           IMPLEMENT YOUR CODE
#
########################################################################
######
        # pass the caption word by word
        for t in range(captions.size(1)):
            # for the first time step the input is the feature vector
            if t == 0:
                features = features[:, :, 0, 0]
                inputs = self.fc_in(features)
                hidden_state, cell_state = self.lstm_cell(inputs, (hidde
n_state, cell_state))

            # for the 2nd+ time step, using teacher forcer
            else:
                hidden_state, cell_state = self.lstm_cell(captions_embed
[:, t, :], (hidden_state, cell_state))

#              print(hidden_state.size())
            # output of the attention mechanism
            out = self.fc_out(hidden_state)
            # build the output tensor
            outputs[:, t, :] = out
########################################################################
```

Problem 2:

- 1.Connect CNN model to the designed LSTM model.
- 2.Replace ResNet to your own CNN model from Assignment3.
- https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning (https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning)

In [6]:

```
########################################################################
######
#                           IMPLEMENT YOUR CODE                        
#
########################################################################
######

# Define a CNN model
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 7)
        self.conv2 = nn.Conv2d(8, 16, 4)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16 * 5 * 5, 100)
        self.fc2 = nn.Linear(100, 80)
        self.fc3 = nn.Linear(80, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# betternet = Net()

"""ResNet"""
#CNN
betternet = resnet.resnet18(pretrained=False)
betternet.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), pa
dding=(3, 3), bias=False)
# betternet.fc = nn.Linear(in_features=512, out_features=ALL_CHAR_SET_LE
N, bias=True)
betternet.fc = nn.Linear(in_features=512, out_features=ALL_CHAR_SET_LEN*
MAX_CAPTCHA, bias=True)
betternet = betternet.to(device)
########################################################################
######
#                           END OF YOUR CODE                           
#
########################################################################
######


# LSTM
cnn_dim=512 #resnet18-512
hidden_size=8
vocab_size=37 #ALL_CHAR_SET_LEN
lstm = LSTM(cnn_dim=cnn_dim, hidden_size=hidden_size, vocab_size=vocab_s
ize)
lstm = lstm.to(device)

# loss, optimizer
########################################################################
######
#                           IMPLEMENT YOUR CODE                        
#
########################################################################
######

class MyEnsemble(nn.Module):
    def __init__(self, modelA, modelB):
        super(MyEnsemble, self).__init__()
        self.modelA = modelA
        self.modelB = modelB
```

Problem3: Find hyper-parameters.

In [7]:

```
import time
import matplotlib.pyplot as plt

"""TRAINING"""
print_interval = 5
max_epoch = 200

x_labels = []
y_labels = []

start_time = time.time()
for epoch in range(max_epoch):
    start_epoch_time = time.time()
    for step, i in enumerate(train_dl):
        start_step_time = time.time()
        img, label_oh, label = i
        img = Variable(img).cuda()
        label_oh = Variable(label_oh.long()).cuda()
########################################################################
######
#                          IMPLEMENT YOUR CODE
#
########################################################################
######
        batch_size, _ = label_oh.shape

#         pred, feature = betternet(img)
#         loss = loss_func(pred, label_oh)
#         cnn_optim.zero_grad()
#         loss.backward()
#         cnn_optim.step()

        pred = model(img, label_oh)
        loss = loss_func(pred, label_oh)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
########################################################################
######
#                          END OF YOUR CODE
#
########################################################################
######
        if (epoch+1)%print_interval == 0:
            print('epoch:', epoch+1, 'step:', step+1, 'loss:', loss.item
())
            y_labels.append(loss.item())
            x_labels.append('epoch:{}, step:{}'.format(epoch+1, step+1))
        if (epoch+1) % 20 == 0:
            torch.save(model, './models/BetterNet_LSTM_epoch{}.pth'.form
at(epoch+1))
    print('>> Epoch', epoch+1, 'elapsed time: {:.2f} sec'.format(time.ti
me()-start_epoch_time))
print('Total Elapsed Time: {:.2f} sec'.format(time.time()-start_time))
```

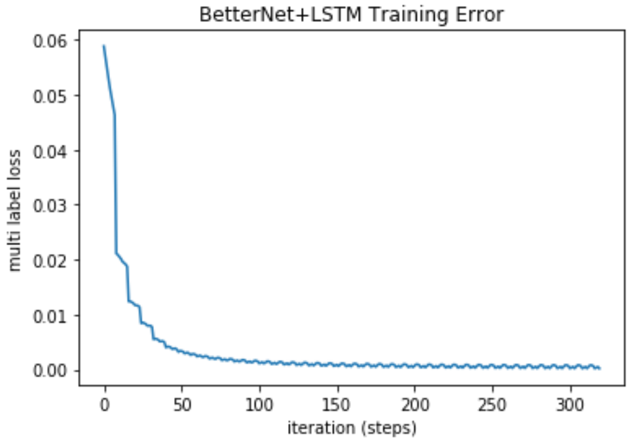
```
>> Epoch 1 elapsed time: 1.70 sec
>> Epoch 2 elapsed time: 1.53 sec
>> Epoch 3 elapsed time: 1.51 sec
>> Epoch 4 elapsed time: 1.55 sec
epoch: 5 step: 1 loss: 0.05884035304188728
epoch: 5 step: 2 loss: 0.05680178850889206
epoch: 5 step: 3 loss: 0.05481883883476257
epoch: 5 step: 4 loss: 0.05292641371488571
epoch: 5 step: 5 loss: 0.05106658488512039
epoch: 5 step: 6 loss: 0.04943714663386345
epoch: 5 step: 7 loss: 0.04787255823612213
epoch: 5 step: 8 loss: 0.046319738030433655
>> Epoch 5 elapsed time: 1.55 sec
>> Epoch 6 elapsed time: 1.47 sec
>> Epoch 7 elapsed time: 1.47 sec
>> Epoch 8 elapsed time: 1.47 sec
>> Epoch 9 elapsed time: 1.48 sec
epoch: 10 step: 1 loss: 0.021118484437465668
epoch: 10 step: 2 loss: 0.020887982100248337
epoch: 10 step: 3 loss: 0.020544061437249184
epoch: 10 step: 4 loss: 0.020159199833869934
epoch: 10 step: 5 loss: 0.019660821184515953
epoch: 10 step: 6 loss: 0.01941952481865883
epoch: 10 step: 7 loss: 0.019157398492097855
epoch: 10 step: 8 loss: 0.01876315474510193
>> Epoch 10 elapsed time: 1.54 sec
>> Epoch 11 elapsed time: 1.47 sec
>> Epoch 12 elapsed time: 1.48 sec
>> Epoch 13 elapsed time: 1.53 sec
>> Epoch 14 elapsed time: 1.48 sec
epoch: 15 step: 1 loss: 0.01239107921719551
epoch: 15 step: 2 loss: 0.012401865795254707
epoch: 15 step: 3 loss: 0.012271185405552387
epoch: 15 step: 4 loss: 0.01207145769149065
epoch: 15 step: 5 loss: 0.011727986857295036
epoch: 15 step: 6 loss: 0.011680996045470238
epoch: 15 step: 7 loss: 0.01162349060177803
epoch: 15 step: 8 loss: 0.01136499922722578
>> Epoch 15 elapsed time: 1.55 sec
>> Epoch 16 elapsed time: 1.51 sec
>> Epoch 17 elapsed time: 1.47 sec
>> Epoch 18 elapsed time: 1.50 sec
>> Epoch 19 elapsed time: 1.50 sec
epoch: 20 step: 1 loss: 0.008435345254838467
```

```
/home/lucetre/anaconda3/lib/python3.7/site-packages/torch/serialization.p
y:402: UserWarning: Couldn't retrieve source code for container of type M
yEnsemble. It won't be checked for correctness upon loading.
  "type " + obj.__name__ + ". It won't be checked "
/home/lucetre/anaconda3/lib/python3.7/site-packages/torch/serialization.p
y:402: UserWarning: Couldn't retrieve source code for container of type L
STM. It won't be checked for correctness upon loading.
  "type " + obj.__name__ + ". It won't be checked "
```

```
epoch: 20 step: 2 loss: 0.008522885851562023
epoch: 20 step: 3 loss: 0.008450414054095745
epoch: 20 step: 4 loss: 0.008295370265841484
epoch: 20 step: 5 loss: 0.007976464927196503
epoch: 20 step: 6 loss: 0.007989149540662766
epoch: 20 step: 7 loss: 0.007995249703526497
epoch: 20 step: 8 loss: 0.007762659806758165
>> Epoch 20 elapsed time: 1.92 sec
>> Epoch 21 elapsed time: 1.51 sec
>> Epoch 22 elapsed time: 1.49 sec
>> Epoch 23 elapsed time: 1.48 sec
>> Epoch 24 elapsed time: 1.50 sec
epoch: 25 step: 1 loss: 0.0055036386474967
epoch: 25 step: 2 loss: 0.005624980200082064
epoch: 25 step: 3 loss: 0.005569618195295334
epoch: 25 step: 4 loss: 0.00542638823390007
epoch: 25 step: 5 loss: 0.005104358308017254
epoch: 25 step: 6 loss: 0.005145411007106304
epoch: 25 step: 7 loss: 0.005182521417737007
epoch: 25 step: 8 loss: 0.004957329016178846
>> Epoch 25 elapsed time: 1.55 sec
>> Epoch 26 elapsed time: 1.48 sec
>> Epoch 27 elapsed time: 1.47 sec
>> Epoch 28 elapsed time: 1.52 sec
>> Epoch 29 elapsed time: 1.52 sec
epoch: 30 step: 1 loss: 0.00402288930490613
epoch: 30 step: 2 loss: 0.004189168103039265
epoch: 30 step: 3 loss: 0.004164414945989847
epoch: 30 step: 4 loss: 0.0040447041392326355
epoch: 30 step: 5 loss: 0.0037328172475099564
epoch: 30 step: 6 loss: 0.003806432243436575
epoch: 30 step: 7 loss: 0.003875649766996503
epoch: 30 step: 8 loss: 0.0036627002991735935
>> Epoch 30 elapsed time: 1.55 sec
>> Epoch 31 elapsed time: 1.47 sec
>> Epoch 32 elapsed time: 1.48 sec
>> Epoch 33 elapsed time: 1.47 sec
>> Epoch 34 elapsed time: 1.46 sec
epoch: 35 step: 1 loss: 0.003203537780791521
epoch: 35 step: 2 loss: 0.0033884630538523197
epoch: 35 step: 3 loss: 0.003372106235474348
epoch: 35 step: 4 loss: 0.0032559200190007687
epoch: 35 step: 5 loss: 0.0029377052560448647
epoch: 35 step: 6 loss: 0.003023972734808922
epoch: 35 step: 7 loss: 0.003106416668742895
epoch: 35 step: 8 loss: 0.002891320502385497
>> Epoch 35 elapsed time: 1.55 sec
>> Epoch 36 elapsed time: 1.50 sec
>> Epoch 37 elapsed time: 1.51 sec
>> Epoch 38 elapsed time: 1.48 sec
>> Epoch 39 elapsed time: 1.49 sec
epoch: 40 step: 1 loss: 0.0026443148963153362
epoch: 40 step: 2 loss: 0.0028423878829926252
epoch: 40 step: 3 loss: 0.0028307773172855377
epoch: 40 step: 4 loss: 0.002715159673243761
epoch: 40 step: 5 loss: 0.002389252418652177
epoch: 40 step: 6 loss: 0.002484065480530262
epoch: 40 step: 7 loss: 0.002575429156422615
epoch: 40 step: 8 loss: 0.002356345299631357
>> Epoch 40 elapsed time: 1.96 sec
>> Epoch 41 elapsed time: 1.46 sec
>> Epoch 42 elapsed time: 1.48 sec
>> Epoch 43 elapsed time: 1.49 sec
>> Epoch 44 elapsed time: 1.49 sec
epoch: 45 step: 1 loss: 0.002240701112896204
epoch: 45 step: 2 loss: 0.0024490910582244396
epoch: 45 step: 3 loss: 0.002440521027892828
epoch: 45 step: 4 loss: 0.0023242677561938763
```

In [8]:
```
plt.plot(y_labels)
plt.xlabel('iteration (steps)')
plt.ylabel('multi label loss')
plt.title('BetterNet+LSTM Training Error')
# plt.show()

plt.savefig('BetterNet+LSTM.png', dpi = 300)
```



In [10]:

```
"""TEST"""
def get_char_count(arg1):
    c0 = ALL_CHAR_SET[np.argmax(arg1.cpu().tolist()[0:ALL_CHAR_SET_LEN])]
    c1 = ALL_CHAR_SET[np.argmax(arg1.cpu().tolist()[ALL_CHAR_SET_LEN:ALL_CHAR_SET_LEN*2])]
    c2 = ALL_CHAR_SET[np.argmax(arg1.cpu().tolist()[ALL_CHAR_SET_LEN*2:ALL_CHAR_SET_LEN*3])]
    c3 = ALL_CHAR_SET[np.argmax(arg1.cpu().tolist()[ALL_CHAR_SET_LEN*3:ALL_CHAR_SET_LEN*4])]
    c4 = ALL_CHAR_SET[np.argmax(arg1.cpu().tolist()[ALL_CHAR_SET_LEN*4:ALL_CHAR_SET_LEN*5])]
    c5 = ALL_CHAR_SET[np.argmax(arg1.cpu().tolist()[ALL_CHAR_SET_LEN*5:ALL_CHAR_SET_LEN*6])]
    c6 = ALL_CHAR_SET[np.argmax(arg1.cpu().tolist()[ALL_CHAR_SET_LEN*6:ALL_CHAR_SET_LEN*7])]
    return c0, c1, c2, c3, c4, c5, c6

def get_str(ch_arr):
    ch_str = ''
    for ch in ch_arr:
        if ch == 'NONE':
            ch_str = ch_str + '_'
        else:
            ch_str = ch_str + ch
    return ch_str

char_correct = 0
word_correct = 0
total = 0

betternet.eval()
lstm.eval()
model.eval()

with torch.no_grad():
    for step, (img, label_oh, label) in enumerate(test_dl):
        char_count = 0
        img = Variable(img).cuda()
        label_oh = Variable(label_oh.long()).cuda()

#          pred, feature = betternet(img)
        pred = model(img, label_oh)

        label_len = label[0]
        pred = pred.squeeze(0)
        label_oh = label_oh.squeeze(0)

        c0,c1,c2,c3,c4,c5,c6 = get_char_count(pred.squeeze())
        d0,d1,d2,d3,d4,d5,d6 = get_char_count(label_oh)

        c_arr = (c0, c1, c2, c3, c4, c5, c6)
        d_arr = (d0, d1, d2, d3, d4, d5, d6)

        c = '%s%s%s%s%s%s%s' % c_arr
        d = '%s%s%s%s%s%s%s' % d_arr

        c_str = get_str(c_arr)
        d_str = get_str(d_arr)

        print('PREDICT:', c_str, ', LABEL:', d_str)

        char_count += (c0==d0)+(c1==d1)+(c2==d2)+(c3==d3)+(c4==d4)+(c5==d5)+(c6==d6)
        char_correct += char_count

        if(bool(str(label[0]) in str(c))):
            word_correct+=1
```

```
PREDICT: b9x____ , LABEL: b9x____
PREDICT: mb_____ , LABEL: mb_____
PREDICT: d5q7qh_ , LABEL: d5q7qh_
PREDICT: 6tl0kqv , LABEL: 6tl0kqv
PREDICT: t1_____ , LABEL: t1_____
PREDICT: avhjn3z , LABEL: avhjn3z
PREDICT: 74z0z__ , LABEL: 74z0z__
PREDICT: f1kfa__ , LABEL: f1kfa__
PREDICT: sripns_ , LABEL: sripns_
PREDICT: bg4____ , LABEL: bg4____
PREDICT: gmb45tz , LABEL: gmb45tz
PREDICT: sr5____ , LABEL: sr5____
PREDICT: 0nt____ , LABEL: 0nt____
PREDICT: lxfg98_ , LABEL: lxfg98_
PREDICT: 2b8o___ , LABEL: 2b8o___
PREDICT: kr25___ , LABEL: kr25___
PREDICT: fl_____ , LABEL: fl_____
PREDICT: 0tiwrd_ , LABEL: 0tiwrd_
PREDICT: k5_____ , LABEL: k5_____
PREDICT: 8k_____ , LABEL: 8k_____
PREDICT: ggin___ , LABEL: ggin___
PREDICT: qc6e___ , LABEL: qc6e___
PREDICT: giz6rv_ , LABEL: giz6rv_
PREDICT: tf15___ , LABEL: tf15___
PREDICT: 7jz____ , LABEL: 7jz____
PREDICT: v3zl9__ , LABEL: v3zl9__
PREDICT: p78ec__ , LABEL: p78ec__
PREDICT: 7rh____ , LABEL: 7rh____
PREDICT: exqo___ , LABEL: exqo___
PREDICT: yrs____ , LABEL: yrs____
PREDICT: si_____ , LABEL: si_____
PREDICT: n4tikm_ , LABEL: n4tikm_
PREDICT: l2c8p__ , LABEL: l2c8p__
PREDICT: fixf___ , LABEL: fixf___
PREDICT: d0j1f__ , LABEL: d0j1f__
PREDICT: higm___ , LABEL: higm___
PREDICT: 3qty___ , LABEL: 3qty___
PREDICT: lnw____ , LABEL: lnw____
PREDICT: kid92__ , LABEL: kid92__
PREDICT: fg_____ , LABEL: fg_____
PREDICT: 7u0____ , LABEL: 7u0____
PREDICT: qbd____ , LABEL: qbd____
PREDICT: ofb____ , LABEL: ofb____
PREDICT: zdn____ , LABEL: zdn____
PREDICT: sqvrc__ , LABEL: sqvrc__
PREDICT: xyx2z1_ , LABEL: xyx2z1_
PREDICT: bcdbx__ , LABEL: bcdbx__
PREDICT: ejt____ , LABEL: ejt____
PREDICT: tl1z___ , LABEL: tl1z___
PREDICT: pn_____ , LABEL: pn_____
PREDICT: 2l_____ , LABEL: 2l_____
PREDICT: 5aip1__ , LABEL: 5aip1__
PREDICT: 8l5pw__ , LABEL: 8l5pw__
PREDICT: q7u____ , LABEL: q7u____
PREDICT: ht1kc5_ , LABEL: ht1kc5_
PREDICT: lpxdssw , LABEL: lpxdssw
PREDICT: hweg1gz , LABEL: hweg1gz
PREDICT: ie_____ , LABEL: ie_____
PREDICT: my_____ , LABEL: my_____
PREDICT: qfcmgzd , LABEL: qfcmgzd
PREDICT: kfozxb_ , LABEL: kfozxb_
PREDICT: ip_____ , LABEL: ip_____
PREDICT: d3a0___ , LABEL: d3a0___
PREDICT: z8359__ , LABEL: z8359__
PREDICT: 8cu____ , LABEL: 8cu____
PREDICT: mjnjd99 , LABEL: mjnjd99
PREDICT: ln_____ , LABEL: ln_____
PREDICT: huet___ , LABEL: huet___
```

```
Out[10]: 'END TEST'
```