0.0.1 How was the dataset collected?

Handwriting samples were collected from about 300 UI students. Each student was asked write one page. These samples were scanned and save as jpg files. Here are two of them.
https://www.dropbox.com/s/6e7ehoelcabs3l7/handwritten_sample2-02.jpg?dl=0
https://www.dropbox.com/s/fc8guzif16w1ire/handwritten_sample3-11.jpg?dl=0

About two thirds of the samples were used to create the trainset, and the remaining were used to create the equations.

0.0.2 How was the trainset created?

```
[1]:
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

path=’C:/Research/DCNN/Competition/handwritten_sample2/’
L=os.listdir(path)
A=cv2.imread(path+L[0])
A.shape

[1]: (1700, 2200, 3)

These color images were changed to gray-scale images

[2]:
A=255-A
A=np.max(A,2)
A=A/np.max(A)
if A.shape[0]<A.shape[1]:
    A=A.T
A.shape

[2]: (2200, 1700)

[3]:
plt.rcParams[‘figure.figsize’] = (20.0, 18.0) # set default size of plots
plt.imshow(A[::1])
plt.show()
```
They were rotated and cropped out and resized

[4]:
```
row1 = np.argmax(np.mean(A[100:220, 100:200], 1)) + 100
col1 = np.argmax(np.mean(A[200:250, 100:200], 0)) + 100
row3 = np.argmax(np.mean(A[2000:, 100:200], 1)) + 2000
```
col3=np.argmax(np.mean(A[2000:100:200],0))+100
row2=np.argmax(np.mean(A[row1-50:row1+50,1480:1650],1))+row1-50
col2=np.argmax(np.mean(A[100:300,1480:1650],0))+1480

width=int(np.ceil(np.sqrt((col2-col1)**2+(row2-row1)**2)))
height=int(np.ceil(np.sqrt((col3-col1)**2+(row3-row1)**2)))
M=np.array([[(row3-row1)/height,(col3-col1)/height],[((row2-row1)/
    width,(col2-col1)/width)])
B=np.zeros((height,width,2))
B[:,0]=np.arange(height).reshape(-1,1)
B[:,1]=np.arange(width).reshape(1,-1)
B=B.dot(M)
C=np.zeros((height,width))
for i in range(height):
    for j in range(width):
        p=row1+int(np.floor(B[i,j,0]))
        q=col1+int(np.floor(B[i,j,1]))
        C[i,j]=A[p,q]
C=cv2.resize(C,(120*10,120*13))
print(C.shape)
plt.imshow(C[:,:-1])
plt.show()

(1560, 1200)
The images were then resized

```
C = cv2.resize(C, (120*10, 120*13))
print(C.shape)
plt.imshow(C[:, -1])
```
Slicing out the symbols
for i in range(13):
    B=C[i*120+3:(i+1)*120-3]
    B=B/np.max(B)
    plt.imshow(B[:,:-1])
    plt.show()
    if i==5:
        Three=B
    if i==11:
        dx=B
Symbols that were far off from the center, were re-centered and resized.
left_idx+=1
else:
    break
	right_idx=A.shape[1]-1
for i in range(5):
    if np.sum(A[:,right_idx])>5:
        right_idx=A.shape[1]-i-1
for i in range(h):
    if np.max(A[:,right_idx])<k:
        right_idx-=1
else:
    break
top_idx=0
for i in range(5):
    if np.sum(A[top_idx,:])>5:
        top_idx=i+1
for i in range(w):
    if np.max(A[top_idx,:])<k:
        top_idx+=1
else:
    break
bottom_idx=A.shape[0]-1
for i in range(5):
    if np.sum(A[bottom_idx,:])>5:
        bottom_idx=A.shape[0]-i-1
for i in range(w):
    if np.max(A[bottom_idx,:])<k:
        bottom_idx-=1
else:
    break
B=A[top_idx:bottom_idx+1,left_idx:right_idx+1]
# B=cv2.resize(B,(64-margin[2]-margin[3],64-margin[0]-margin[1]))
# C=np.zeros((64,64))
# C[margin[0]:-margin[1],margin[2]:-margin[3]]=B
# C=C/np.max(C)
return B

[10]: for j in range(10):
    E=center_crop(dx[:,120*j:120*(j+1)-3],(3,3,3,3),0.5)
    print(E.shape)
    plt.subplot(10,10,j+1)
    plt.axis('off')
    plt.imshow(E[:,:,1])
plt.show()
They were resized to the same height of 64, but a varying width between 61 and 67 to reflect the original shape.

```python
[11]: for j in range(10):
    E=center_crop(dx[:,120*j+3:120*(j+1)-3],(3,3,3,3),0.5)
    E=cv2.resize(E,(E.shape[1],64))
    plt.subplot(10,10,j+1)
    plt.axis('off')
    plt.imshow(E[:,:,1])
plt.show()
```

When create trainset, they were packed into a square, and then the reshape the square to the size of 64x64

```python
[12]: for j in range(10):
    E=center_crop(dx[:,120*j+3:120*(j+1)-3],(3,3,3,3),0.5)
    E=cv2.resize(E,(E.shape[1],64))
    W=np.zeros((64,64))
    a=int(32-E.shape[1]/2)
    W[:,:,a:E.shape[1]]=E
    plt.subplot(10,10,j+1)
    plt.axis('off')
    plt.imshow(W[:,:,1])
plt.show()
```
When create the problem equations, the varying width 61-67 were used, except the last symbol. The width of the last symbol was chosen so that the entire equation would have the same length.

### 0.0.3 Baseline Model

Download the train data and train lable and competition Problems and save it to the local directory. In my case, I save them as:

```
'C:/Research/DCNN/Competition/Downloaded_Competition_Train_data_8000.npy'
'C:/Research/DCNN/Competition/Downloaded_Competition_Train_label_8000.npy'
'C:/Research/DCNN/Competition/Downloaded_Competition_Problems.npy'
```

```python
[13]: X=np.load('C:/Research/DCNN/Competition/Downloaded_Competition_Train_data_8000.npy')
y=np.load('C:/Research/DCNN/Competition/Downloaded_Competition_Train_label_8000.npy')
Problems=np.load('C:/Research/DCNN/Competition/Downloaded_Competition_Problems.npy')
print(X.shape)
print(y.shape)
print(Problems.shape)
```

(8000, 64, 64)
(8000,)
(20000, 64, 384)

```python
[14]: import torch
import torch.nn as nn
import torch.optim as optim

# Build a simple model
class baseline_linear(nn.Module):
    def __init__(self):
        super(baseline_linear, self).__init__()
        self.linears=nn.Sequential(nn.Linear(64*64,500), nn.BatchNorm1d(500),
            nn.ReLU(),
            nn.Linear(500, 100), nn.BatchNorm1d(100),nn.
            ReLU(),
            nn.Linear(100,16))
    def forward(self, input):
        x = self.linears(input)
```
def train(epoch, to_print=True):
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs, targets) in enumerate(trainloader):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()
    if to_print:
        print('epoch %d Train loss: %.3f | Acc: %.3f%% (%d/%d)
              % (epoch+1, train_loss/(batch_idx+1), 100.*correct/total, correct, total))

def test(epoch):
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(testloader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)
            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
    print('Test loss: %.3f | Acc: %.3f%% (%d/%d)
          % (test_loss/(batch_idx+1), 100.*correct/total, correct, total))

# the following 'train' and 'test' are what I may use for every model
# Since the model we built above treat each symbol as a 64x64-dimensional vector
# We need to reshape X from (8000,64,64) to 8000, 64*64
X=X.reshape(-1,64*64)
#X=X.reshape(-1,1,64,64)
I=np.arange(X.shape[0])
np.random.shuffle(I)
X_train=X[I[:6000]]
y_train=y[I[:6000]]
X_val=X[I[6000:]]
y_val=y[I[6000:]]

from torch.utils.data import Dataset, DataLoader
class get_data(Dataset):
    def __init__(self, data, label):
        self.data = torch.FloatTensor(data.astype('float'))
        self.label = torch.from_numpy(label).long()

    def __len__(self):
        return self.label.shape[0]

    def __getitem__(self, index):
        data_val = self.data[index]
        target = self.label[index]
        return data_val, target

batch_size = 100
trainset = get_data(X_train, y_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True, num_workers=0)
testset = get_data(X_val, y_val)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=0)

net = baseline_linear()
device = 'cuda' if torch.cuda.is_available() else 'cpu'
if device == 'cuda':
    net = net.cuda()
print("using gpu")
criterion = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(net.parameters(), lr=0.001)

for epoch in range(3):
    train(epoch)
    test(epoch)

# A lazy way of slicing out the symbols: Assume they are of the same size.
# Then only three of the six symbols are relevant, mainly, the second, the fifth, and the last
# Better result can be obtained if I segment the symbols more carefully.
First=Problems[:,:,:,64:128]
Second=Problems[:,:,:,4*64:5*64]
Third=Problems[:,:,:,5*64:]

# predict the cut-off symbols using the trained model
First=np.reshape(First,(-1,1,64*64))
Second=np.reshape(Second,(-1,1,64*64))

#First=np.reshape(First,(-1,1,64,64))
#Second=np.reshape(Second,(-1,1,64,64))
pred_First=np.zeros(First.shape[0])
pred_Second=np.zeros(Second.shape[0])
for i in range(100):
    syms=torch.FloatTensor(First[i*200:(i+1)*200].astype('float')).to(device)
    T=net(syms).detach().cpu().numpy()
    pred_First[i*200:(i+1)*200]=np.argmax(T,1)
    syms=torch.FloatTensor(Second[i*200:(i+1)*200].astype('float')).to(device)
    T=net(syms).detach().cpu().numpy()
    pred_Second[i*200:(i+1)*200]=np.argmax(T,1)

# computing the score according to the grading rules
A=(pred_First<4)
B=(pred_First<7)*(pred_First>3)
C=(pred_First>0)*A
D7=(pred_Second-pred_First==7)
D3=(pred_Second-pred_First==3)
D12=(pred_Second-pred_First==12)
D_3=(pred_Second-pred_First==3)
Score=(A*D7*10+B*D_3*10+A*D12*5+C*D3*2)*(np.mean(np.mean(Third[:,:,:,10:64],-1),-1)>0.03)
Score += (A*D7*9+B*D_3*9+A*D12*5+C*D3*1)*(np.mean(np.mean(Third[:,:,:,10:64],-1),-1)<0.03)

# Save the score as my_submission, which can be uploaded for the submission.
np.save('my_submission.npy',Score)
print('Done! The first 10 problems have predicted scores: ', Score[:10])

using gpu
epoch 1 Train loss: 1.157 | Acc: 67.600% (4056/6000)
Test loss: 0.807 | Acc: 74.700% (1494/2000)
epoch 2 Train loss: 0.645 | Acc: 80.033% (4802/6000)
Test loss: 0.638 | Acc: 78.700% (1574/2000)
epoch 3 Train loss: 0.432 | Acc: 87.000% (5220/6000)
Test loss: 0.594 | Acc: 79.550% (1591/2000)
Done! The first 10 problems have predicted scores: [ 9 1 0 0 0 10 1 0 10 1]