Computer Vision
16720-B Fall 2021

16720 (B) Neural Networks for Recognition - Assignment 3

Instructor: Kris Kitani TAs: Qichen(Lead), Paritosh, Rawal, Yan, Zen, Wen-Hsuan

Submission Instructions:

1. Submit the PDF version of `theory.ipynb` to HW3:PDF. The `theory.ipynb` should include **ALL the writeup answers AND ALL the screenshots of code** specifically required in questions **from Q1 to Q7**. This section will be manually Graded.
2. Submit `q2.ipynb` , `q3.ipynb` , `q5.ipynb` to HW3:Code. Please do not submit other jupyter notebooks as they will not be autograded. Submitting them may cause running time out. (`q5.ipynb` is optional for extra credits)

The Appendix section at the end of this file would help you on questions P1 and P2.

Q1 Theory Questions (45 points)

Q1.1 (4 Points WriteUp)

Prove that softmax is invariant to translation, that is

$$softmax(x) = softmax(x + c) \qquad \forall c \in \mathbb{R}$$

Softmax is defined as below, for each index i in a vector x.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Often we use $c = -\max x_i$. Why is that a good idea? (Tip: consider the range of values that numerator will have with $c = 0$ and $c = -\max x_i$)

Soln 1.1.

$$f(x_i) = \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$f(x_i + c) = \frac{e^{x_i + c}}{\sum_j e^{x_j + c}}$$

$$= \frac{e^{c} e^{x_i}}{\sum_j e^{c} e^{x_j}}$$

$$= f(x_i).$$

Thus $f(x_i) = f(x_i + c)$.

The often use of $c = -\max x_i$ is because.

when $c = -\max x_i$; $x_i + c \leq 0$.

$\therefore \quad e^{x_i + c} \leq 1.$

This ~~prev~~ value of c bounds the function between 0 & 1 and this prevents overflow.

Q1.2

Softmax can be written as a three step processes, with $s_i = e^{x_i}$, $S = \sum_i s_i$ and $softmax(x_i) = \frac{1}{S} s_i$.

Q1.2.1 (1 point WriteUp)

As $x \in \mathbb{R}^d$, what are the properties of $softmax(x)$, namely what is the range of each element? What is the sum over all elements?

Soln 1.2

Soln 1.2).

since $softmax(x) = \frac{s_i}{\Sigma s_i}$; $s_i = e^{x_i}$

The range will be from $[0, 1]$

& sum of all elements $= \Sigma \frac{s_i}{(\Sigma s_i)} \rightarrow$ cond

$= 1$

Q1.2.2 (1 point WriteUp)

One could say that "softmax takes an arbitrary real valued vector x and turns it into a _". **Please think about a short phrase to fill in _.**

Probabilistic distribution

Q1.2.3 (1 point WriteUp)

Can you see the role of each step in the multi-step process now? Explain them.

Yes, First step represents scaled version of each element, second step represents scaled version of all the elements and final steps provides relative representation of each element with respect to whole.

Q1.3 (3 points WriteUp)

Show that multi-layer neural networks without a non-linear activation function are equivalent to linear regression.

Soln 1.3

In the neural network, the function used is

$$x^{k+1} = W_k x^k + b_k.$$

without a nonlinear activation layer the next iteration is as follows.

$$x^{k+2} = W_{k+1}(x^{k+1}) + b_{k+1}$$

$$= W_{k+1}(W_k x^k + b_k) + b_{k+1}$$

$$= W_{k+1} W_k x^k + W_{k+1} b_k + b_{k+1}.$$

This is again a linear function.
Hence it is same as solving linear regression.

Q1.4 (4 points WriteUp)

Given the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$, derive the gradient of the sigmoid function and show that it can be written as a function of $\sigma(x)$ (without having access to x directly)

Soln 1.4: Given $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\frac{d\,\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}}$$

$$= \sigma(x)\left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= \sigma(x)(1-\sigma(x))$$

Q1.5 (12 points WriteUp)

Given $y = Wx + b$ (or $y_j = \sum_{i=1}^{d} x_i W_{ji} + b_j$), and the gradient of some loss J with respect y, show how to get $\frac{\partial J}{\partial W}$, $\frac{\partial J}{\partial x}$ and $\frac{\partial J}{\partial b}$. Be sure to do the derivatives with scalars and re-form the matrix form afterwards. Here are some notional suggestions.

$$\frac{\partial J}{\partial y} = \delta \in \mathbb{R}^{k\times 1} \quad W \in \mathbb{R}^{k\times d} \quad x \in \mathbb{R}^{d\times 1} \quad b \in \mathbb{R}^{k\times 1}$$

Soln 1.5: Given, $y_j = \sum_{i=1}^{n} x_i w_{ij} + b_j$

$$\frac{\partial y_j}{\partial w_{ij}} = x_i, \quad \frac{\partial y_j}{\partial x_i} = w_{ij}$$

$$\frac{\partial y_j}{\partial b_j} = 1 \in R^{N\times 1}; \quad \frac{\partial J}{\partial y_i} = \delta \in R^{N\times C}$$

According to the dimensions in Q2. $(y = xw + b)$

$N \rightarrow$ Dimensionality of data

$C \rightarrow$ number of classes

$D \rightarrow$ No: of data set

$H \rightarrow$ Hidden layer

$$x_i \in R^{N\times D}$$

$$w_{ij} \in R^{D\times C}$$

Using chain rule,

$$\partial J/\partial w = \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial w} = x^T \delta \in R^{D\times C}$$

$$\partial J/\partial x = \partial J/\partial y \; \partial y/\partial x = \delta w^T \in R^{N\times D}$$

$$\frac{\partial J}{\partial b} = \sum_{i=1}^{N} \delta_i \in R^{1\times C}$$

For the form, $\underset{(C\times N)}{y} = \underset{(C\times D)}{w^T}\underset{(D\times N)}{x} + \underset{(C\times 1)}{b}$,

the equation will be.

$$\frac{\partial J}{\partial w} = x \delta^T \in R^{C\times D} \left[\delta \in R^{C\times N}\right]$$

$\partial w \quad (D\times N)(N\times C) \quad D\times N$

$$\frac{\partial J}{\partial x} = W\delta \in R$$

$$\partial J / \partial b = \sum_{l=1}^{N} \delta \in R^{C\times 1}$$

Q1.6 (15 points WriteUp)

We will find the derivatives for Conv layers now. Since most Deep Learning frameworks such as Pytorch, Tensorflow use cross-correlation in their respective "convolution" functions ([Pytorch (https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d)](https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d) and [Tensorflow (https://www.tensorflow.org/api_docs/python/tf/nn/convolution)](https://www.tensorflow.org/api_docs/python/tf/nn/convolution)), we will continue this abuse of notation. So the operation performed with the Conv Layer weights will be cross-correlation.

The input, x is of shape $M \times N$ with C channels. This will be *convolved* (actually cross-correlation) with D number of $K \times K$ filters, each with a bias term. The stride is 1 and there will be no padding. We know the gradient of some loss J with respect to the output y, which will have D channels. Show how to get $\frac{\partial J}{\partial W}$, $\frac{\partial J}{\partial x}$ and $\frac{\partial J}{\partial b}$.

The dimensions and notation are as follows:

$$\frac{\partial J}{\partial y} = \delta \in \mathbb{R}^{D\times M_o\times N_o} \quad M_o = M - K + 1 \quad N_o = N - K + 1$$

$$x \in \mathbb{R}^{C\times M\times N} \quad W \in \mathbb{R}^{D\times C\times K\times K} \quad b \in \mathbb{R}^{D}$$

$x_{c,i,j}$: The element at the ith row, the jth column and the cth channel of the input

$y_{c,i,j}$: The element at the ith row, the jth column and the cth channel of the output

$W_{d,c,i,j}$: The element at the ith row, the jth column, the cth channel of the kernel of the dth filter

For this question, you may compute the derivatives with scalars only. You don't need to re-form the matrix

1.6 From the forward pass equation we have

$$y_{dij} = \sum_{l=1}^{C} \sum_{m=1}^{K} \sum_{n=1}^{K} x(l, m+i, n+j)\, w(d,l,m,n) + b_d.$$

$\Rightarrow$ Cross correlation formula.

$$\frac{\partial y_{dij}}{\partial w_{dcij}} = x(l, m+i, n+j)$$

Given $\frac{\partial J}{\partial y_{dij}} = \delta_{dmn} \in R^{D \times M_0 \times N_0}$. $\delta_{dmn} \in R^{D}$, $d \in R^{M_0}$, $m \in R^{N_0}$

$$\therefore \frac{\partial J}{\partial w_{dcij}} = \frac{\partial J}{\partial y_{dij}} \times \frac{\partial y_{dij}}{\partial w_{dcij}}$$

$$= \sum_{d=1}^{D} \sum_{l=1}^{C} \sum_{m=1}^{M_0} \sum_{n=1}^{N_0} x(l, m+i, n+j)\, \delta_{dmn}.$$

Here the d & l term doesn't contribute to the chain rule.

$$\boxed{\therefore \frac{\partial J}{\partial w_{dcij}} = \sum_{m=1}^{M_0} \sum_{n=1}^{N_0} x(c, m+i, n+j)\, \delta_{dmn}}$$

We have,

$$y_{dij} = \sum_{l=1}^{C} \sum_{m=1}^{K} \sum_{n=1}^{K} x(c, m+i, n+j)\, w(d, l, m, n) + b_d.$$

$$\frac{\partial J}{\partial x_{cij}} = \sum_{d}^{D} \sum_{l}^{C} \sum_{i}^{M_0} \sum_{j}^{N_0} \frac{\partial J}{\partial y_{dij}} \frac{\partial y_{dij}}{\partial x_{cij}}$$

$$\frac{\partial y_{dij}}{\partial x_{cij}} = \sum_{l}^{C} \sum_{m}^{M_0} \sum_{n}^{N_0} w(d, l, m, n)$$

$$\frac{\partial J}{\partial y_{dij}} = \sum_{m}^{D} \sum_{n}^{M_0} \sum_{d}^{N_0} \delta_{dij}\, \delta_{dmn} \in R^{D \times M_0 \times N_0}$$

$$d \in R^{D}, m \in R^{M_0}, n \in R^{N_0}$$

$$\therefore \frac{\partial J}{\partial x_{cij}} = \sum_{d=1}^{D} \sum_{l=1}^{C} \sum_{m=1}^{M_0} \sum_{n=1}^{N_0} w(d, l, m, n)\, \delta_{dmn}.$$

Here the term l doesn't contribute to the chain rule

$$\boxed{\therefore \frac{\partial J}{\partial x_{cij}} = \sum_{d=1}^{D} \sum_{m=1}^{M_0} \sum_{n=1}^{N_0} w(d, c, m, n)\, \delta_{dmn}.}$$

$$\frac{\partial J}{\partial b_d} = \frac{\partial J}{\partial y_{dij}} = \frac{\partial y_{dij}}{\partial b_d}.$$

$$\frac{\partial y_{dij}}{\partial b_d} = \sum_{m=1}^{M_0} \sum_{n=1}^{N_0} 1$$

$$\therefore \frac{\partial J}{\partial b_d} = \sum_{d=1}^{D} \sum_{m=1}^{M_0} \sum_{n=1}^{N_0} \delta_{dijmn} \times 1.$$

Here d doesn't contribute in multiplication

$$\therefore \frac{\partial J}{\partial b_d} = \sum_{m=1}^{M_0} \sum_{n=1}^{N_0} \delta_{dmn}.$$

Q1.7

When the neural network applies the elementwise activation function (such as sigmoid), the gradient of the activation function scales the back-propagation update. This is directly from the chain rule, $\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$.

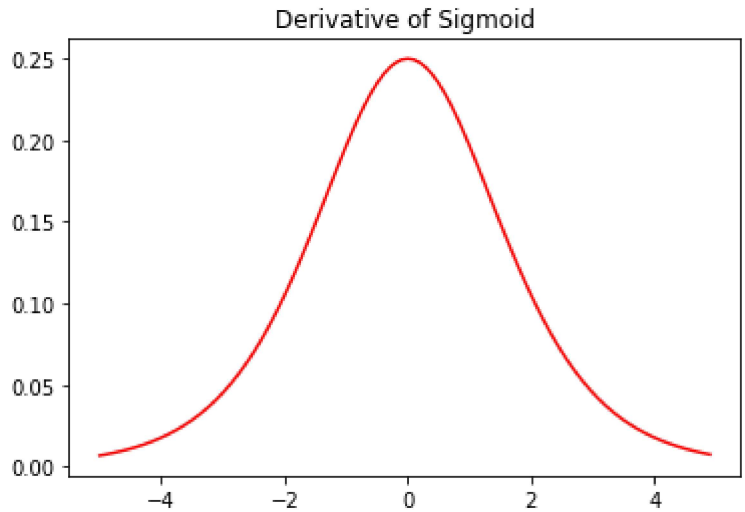
Q1.7.1 (1 point WriteUp)

Consider the sigmoid activation function for deep neural networks. Why might it lead to a "vanishing gradient" problem if it is used for many layers (consider plotting the $\sigma'(x)$ in Q1.4)?

The derivative of sigmoid is as shown below: When we apply this function during backprop, as its value ranges from(0,0.25), each time when this fucnction is used, it will scale down the gradients. So, when this is used multiple times it will diminish the gradient value nearly to zero, hence causing vanishing gradient issue.

In [16]:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5, 5, 0.1)
y =np.exp(-x) /(1+np.exp(-x))**2
plt.plot(x,y, color ='r', label ='tanh')
plt.title('Derivative of Sigmoid')
plt.show()
```



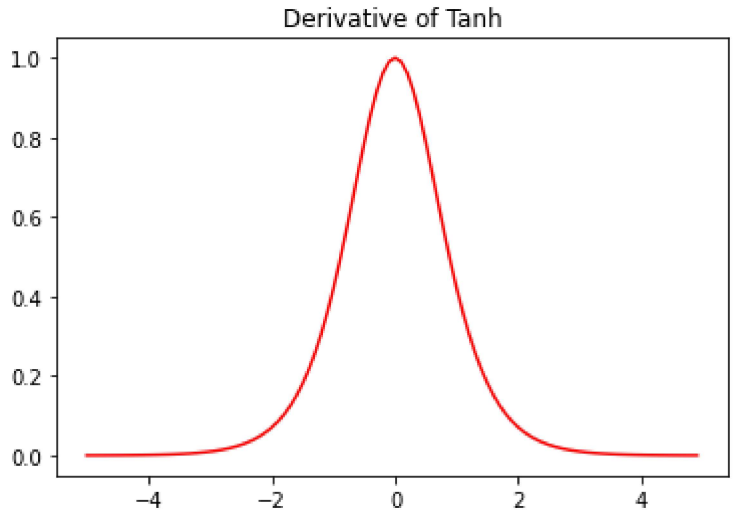
Q1.7.2 (1 point WriteUp)

Often it is replaced with $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$. What are the output ranges of both $\tanh$ and sigmoid? Why might we prefer $\tanh$?

The output range of derivative of tanh and sigmoid output ranges is from (0,1). We prefer tanh because the derivative of tanh is from (0, 1) unlike sigmoid. Thus it doesnt diminish the gradient as fast as sigmoid. Also, tanhx mean is centered at zero, so it converges faster, which makes it easier to train the model

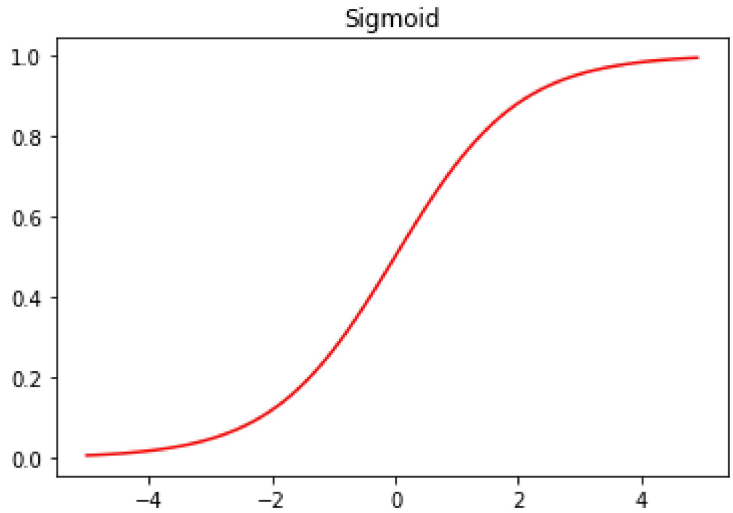
In [11]:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5, 5, 0.1)
y =1- np.tanh(x)**2
plt.plot(x,y, color ='r', label ='tanh')
plt.title('Derivative of Tanh')
plt.show()
```



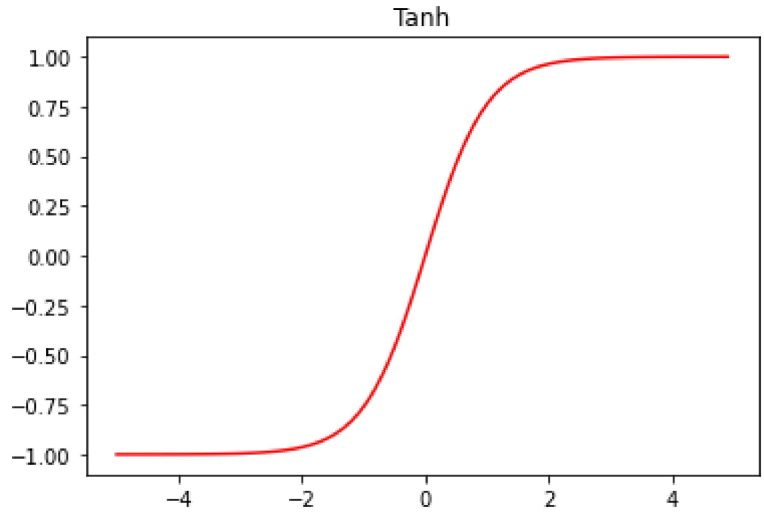
In [13]:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5, 5, 0.1)
y =1 /(1+np.exp(-x))
plt.plot(x,y, color ='r', label ='tanh')
plt.title('Sigmoid')
plt.show()
```



In [9]:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5, 5, 0.1)
y =(1 - np.exp(-2*x))/(1+np.exp(-2*x))
plt.plot(x,y, color ='r', label ='tanh')
plt.title('Tanh')
plt.show()
```



Q1.7.3 (1 point WriteUp)

Why does $\tanh(x)$ have less of a vanishing gradient problem? (plotting the derivatives helps! for reference: $\tanh'(x) = 1 - \tanh(x)^2$)

As shown in the above graph, the range of derivative of tanh is from (0,1) and of sigmoid is from (0, 0.25). So,

derivative of tanh has lesser gradient vanishing probelm when compared to derivative of sigmoid.

Q1.7.4 (1 point WriteUp)

$\tanh$ is a scaled and shifted version of the sigmoid. Show how $\tanh(x)$ can be written in terms of $\sigma(x)$. (*Hint: consider how to make it have the same range*)

σ(2x) = 1/(1 + exp(-2x))

2σ(2x) -1 = 2/(1 + exp(-2x)) -1 = (1 - exp(-2x))/(1 + exp(-2x)) = tanhx

tanhx=2σ(2x)−1

For the following questions, please find the instructions in the corresponding jupyter notebooks.

Q2 Implement a Fully Connected Network (65 points + 10 Extra Credit)

Q2.1.1 (3 points WriteUp)

If the weights and bias is set zero during the initialization, during the forward propagation, each hidden node wouldnt get an input (zero input) from input layer. During the back propagation, the due tho zero weights, it will give same gradients and thus do the saame computation. This is not good for optimal training.

Q2.1.3 (2 points WriteUp)

Random initialization provides uniform data distribution in the matrix and prevent same computation in the network. Thus, it increases the chance of reaching optimal parameters faster which can lead to low loss.

Scaling initalization based on layer size will keep the variance of the gradients of parameters same (same order) for all the layers. Thus, during forward propagation and back propagation, it will prevent gradient explosion and vanishing gradient problem.

Q3 Training Models (20 Points)

Q3.2 (3 points Code+WriteUp)

The figure below shows the train and test acuracy and loss for 3 learning rate. The optimal learning rate is 2.55e-3 and the optimal accuracy in test set is 75.11%.

Comment on learning rate:

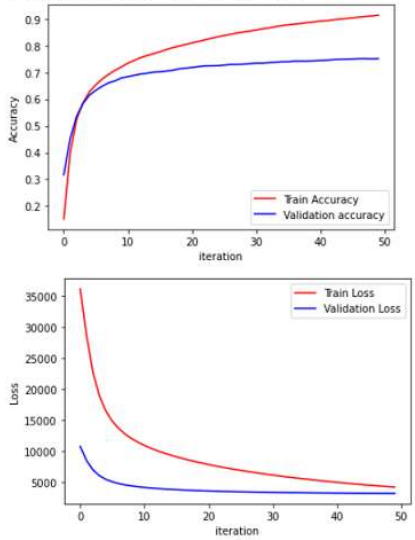
With optimal learning rate we obtain the max accuracy. On decreasing the learning rate, the curve will be smooth but it leanrs slowly and thus it never reaches the optimal accuracy. When we increase the learning rate, the variation of the curve (alot of spikes) will increase and thus it has more purturbation and wont converge properly

```
learning_rate = 2.55e-3

NN(learning_rate)
```

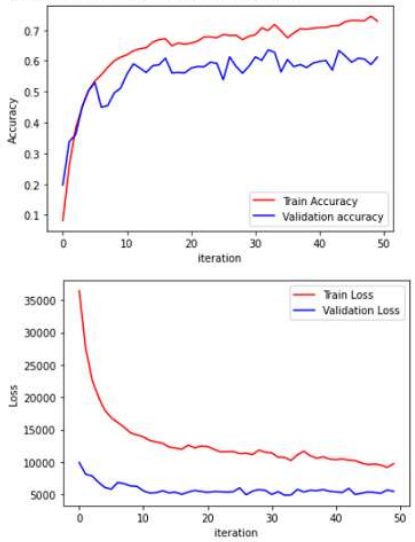
```
[18]: learning_rate = 2.55e-2
NN(learning_rate)
```

```
learning_rate = 2.55e-4
NN(learning_rate)
```

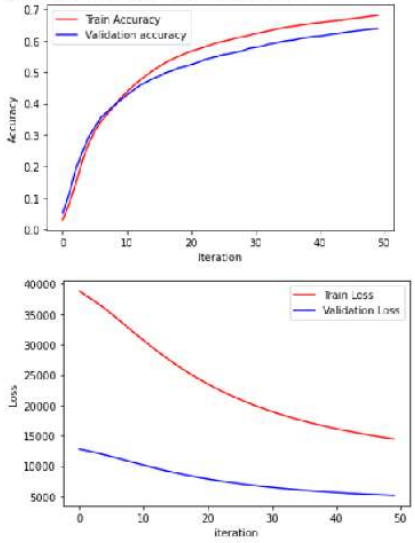
Validation accuracy: 0.7511111111111111



Validation accuracy: 0.6122222222222222



Validation accuracy: 0.6386111111111111



```
#load the dataset
train_data = scipy.io.loadmat('data/nist36_train.mat')
valid_data = scipy.io.loadmat('data/nist36_valid.mat')

train_x, train_y = train_data['train_data'], train_data['train_labels']
valid_x, valid_y = valid_data['valid_data'], valid_data['valid_labels']

#initial parameters
max_iters = 80
# pick a batch size, learning rate

batch_size = 30
learning_rate = 2.55e-3
hidden_size = 64
```

```
#Get batches
batches = get_random_batches(train_x,train_y,batch_size)
batch_num = len(batches)
valid_acc = None
params = {}

# size of weight is inputlayer size x output layer size  ::: x -- N X D and Y --- N x C
initialize_weights(train_x.shape[1],hidden_size,params,'layer1')
initialize_weights(hidden_size,train_y.shape[1],params,'output')
# initialize layers (named "layer1" and "output") here

#Initialization for the list for appending loss and accuracy
train_acclist =[]
valid_acclist= []
train_losslist =[]
valid_losslist= []

# with default settings, you should get loss < 150 and accuracy > 80%

for itr in range(max_iters):
    total_loss = 0
    total_acc = 0
    avg_acc =0
    #For Training the datasets
    for xb,yb in batches:

        #Forward Propagation
```

```
        #Forward Propagation
        post_act1 =forward(xb, params, 'layer1',sigmoid)
        post_act2 = forward(post_act1,params,'output',softmax)

        #Finding the loss and accuracy
        loss , accuracy = compute_loss_and_acc(yb, post_act2)

        #adding the loss and accuracy to epoch totals
        total_loss += loss
        total_acc += accuracy


        delta1 =  post_act2 - yb
        #Finding delta 1

        # backward prop
        delta2 = backwards(delta1,params,'output',linear_deriv)
        grad_X = backwards(delta2,params,'layer1',sigmoid_deriv)

        #updating the parameters
        params['W' + 'output'] -= (learning_rate)*params['grad_W' + 'output']
        params['b' + 'output'] -= (learning_rate)*params['grad_b' + 'output']
        params['W' + 'layer1'] -= (learning_rate)*params['grad_W' + 'layer1']
        params['b' + 'layer1'] -= (learning_rate)*params['grad_b' + 'layer1']

    # Calculating avg accuracy and loss
    avg_acc = total_acc/batch_num
    total_loss = total_loss/batch_num
    if itr % 2 == 0:

        print("itr: {:02d} \t loss: {:.2f} \t acc : {:.2f} \t avg_acc : {:.2f}".format(
            itr,total_loss,total_acc, avg_acc))

# run on validation set and report accuracy! should be above 75%

    #appending the accuracy and loss
    train_acclist.append(avg_acc)
    train_losslist.append(total_loss)

    #Performing Validation
    #Calculating the forward propagation for validation
    valid_post_act1 =forward(valid_x, params, 'layer1',sigmoid)
    valid_post_act2 = forward(valid_post_act1,params,'output',softmax)

    #Calculating the loss and accuracy for validation
    valid_loss , valid_acc= compute_loss_and_acc(valid_y, valid_post_act2)
```

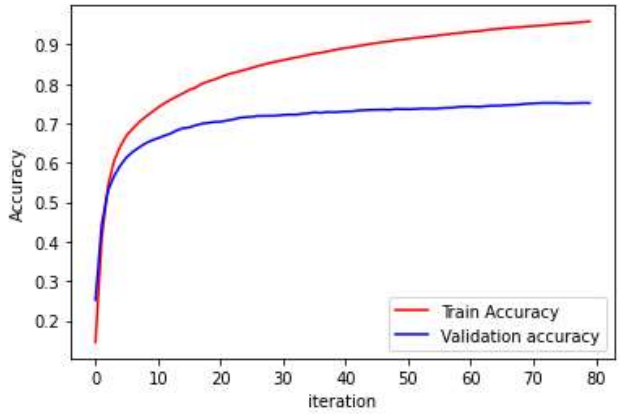
```
    valid_post_act2 = forward(valid_post_act1,params, 'output',softmax)

    #Calculating the loss and accuracy for validation
    valid_loss , valid_acc= compute_loss_and_acc(valid_y, valid_post_act2)

    #Appending them into the list
    valid_acclist.append(valid_acc)
    valid_losslist.append(valid_loss)
    print("valid acc : {:.2f} \t ".format(valid_acc))

print('Validation accuracy: ',valid_acc)
```

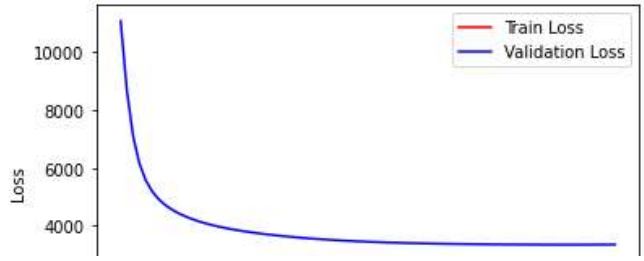
```
valid acc : 0.75
valid acc : 0.75
itr: 74          loss: 7.24      acc : 343.73    avg_acc : 0.95
valid acc : 0.75
valid acc : 0.75
itr: 76          loss: 7.01      acc : 344.47    avg_acc : 0.95
valid acc : 0.75
valid acc : 0.75
itr: 78          loss: 6.79      acc : 345.50    avg_acc : 0.96
valid acc : 0.75
valid acc : 0.75
Validation accuracy:  0.7516666666666667
```

```
#Saving the parameters through pickle
import pickle
# Creating the dictionary
sav_params = {k:v for k,v in params.items() if '_' not in k}
#Saving the weights
with open('q3_weights.pickle', 'wb') as handle:
    pickle.dump(sav_params, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
#Plotting the accuracy and loss for train and validation
a = np.arange(0, max_iters)
train_acclist =np.asarray(train_acclist)
valid_acclist= np.asarray(valid_acclist)
train_losslist =np.asarray(train_losslist)
valid_losslist= np.asarray(valid_losslist)

#print(train_acclist.shape, a.shape)

#For figure 1 - Accuracy
plt.figure(0)
plt.plot(a, train_acclist, 'r')
plt.plot(a, valid_acclist, 'b')
plt.legend(['Train Accuracy', 'Validation accuracy'])
plt.xlabel('iteration')
```

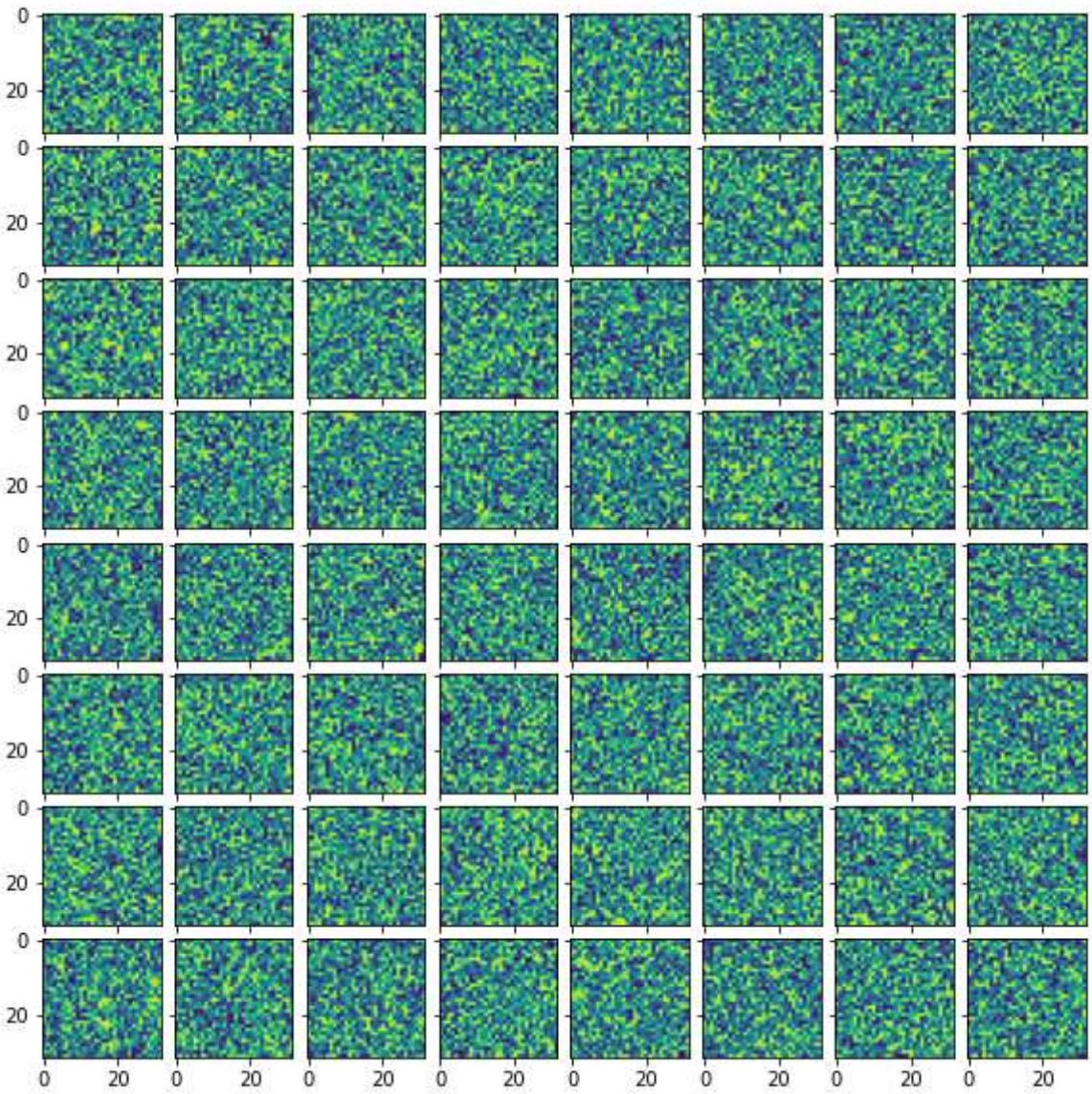
```
#For figure 1 - Accuracy
plt.figure(0)
plt.plot(a, train_acclist, 'r')
plt.plot(a, valid_acclist, 'b')
plt.legend(['Train Accuracy', 'Validation accuracy'])
plt.xlabel('iteration')
plt.ylabel('Accuracy')

#For figure 2 - Loss
plt.figure(1)
plt.plot(a, train_losslist, 'r')
plt.plot(a, valid_losslist, 'b')
plt.legend(['Train Loss', 'Validation Loss'])
plt.xlabel('iteration')
plt.ylabel('Loss')

plt.show()
```



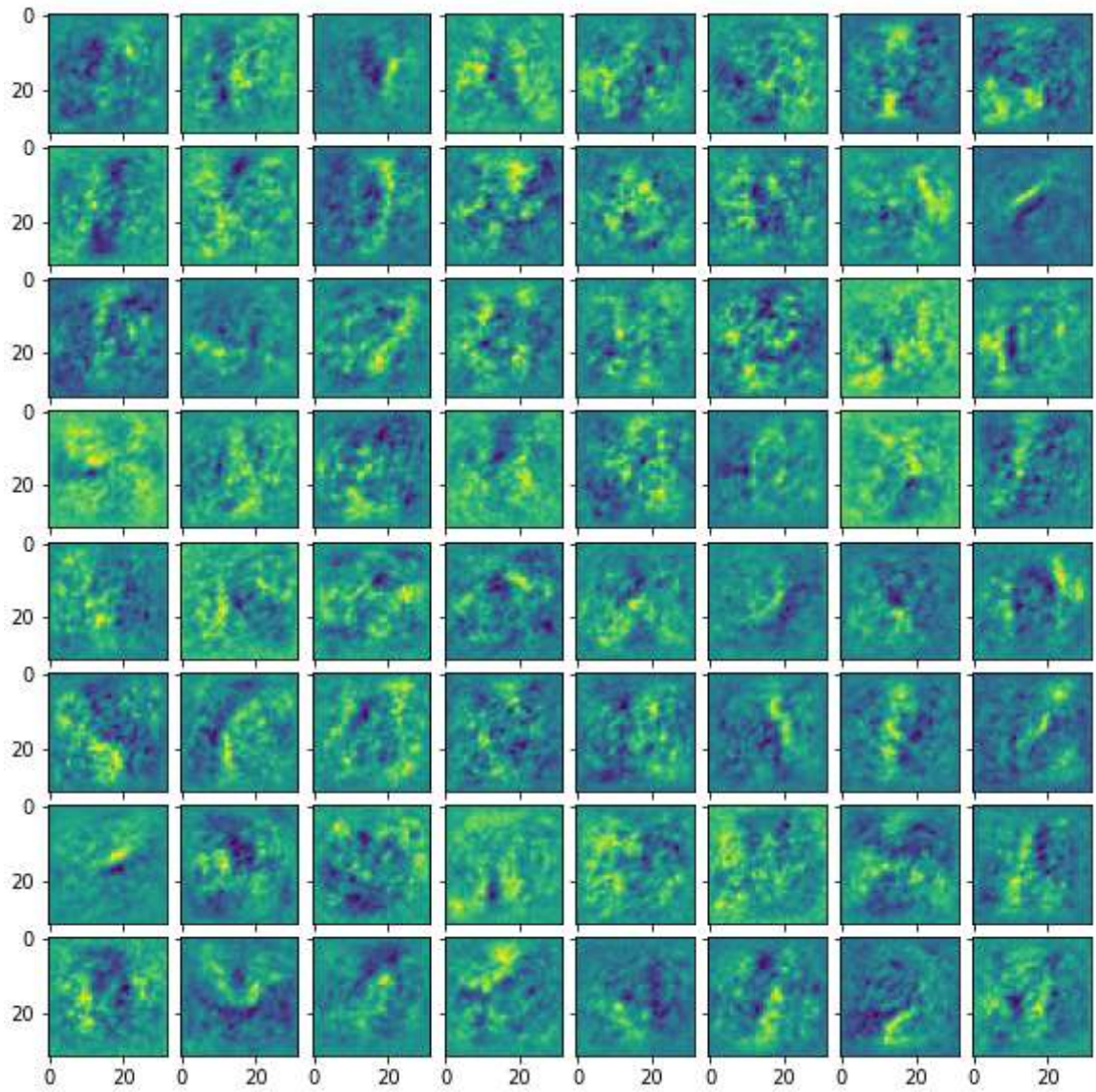


Q3.3 (2 points Code+WriteUp)

Visualization of the weights for the first layer just after initialization is as follows



Visualization of the weights for the first layer after learning is as follows:



Comparing these two figures, we can see the weights just after initialization shows random pattern, meanwhile the one which is learned shows clearer pattern.

Code for learned weights

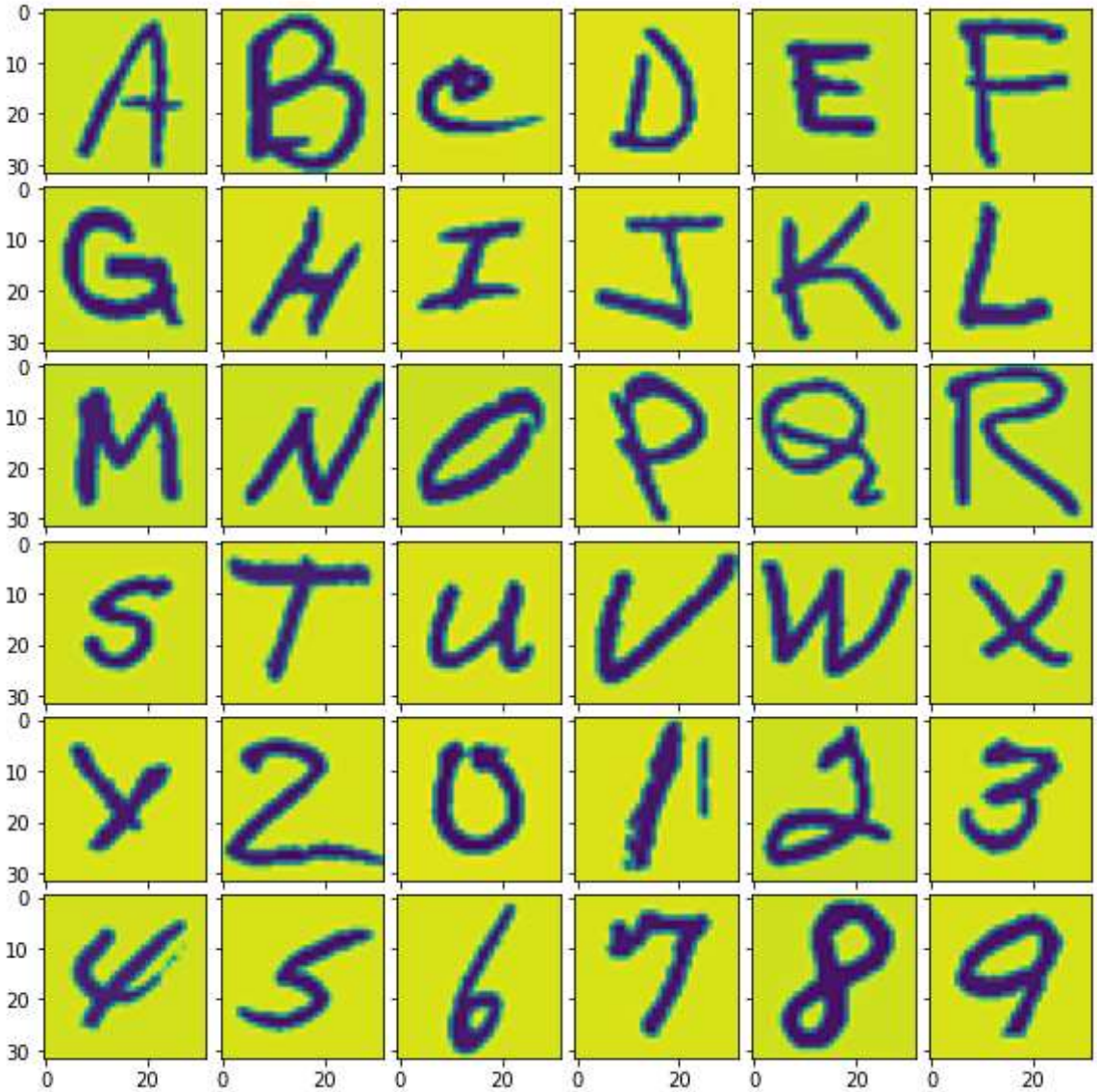
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid

#As becomes a matrix Wx with output size 64 elements. Drawing it as a 8x8 image, since getting 1024, as invalid
# reshaping equal dimentions
fig = plt.figure(figsize=(10, 10))

#Creating a grid for plotting the image
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(8, 8),  # creates 2x2 grid of axes
                 axes_pad=0.1,  # pad between axes in inch.
                 )


m = int(np.sqrt(params['Wlayer1'].shape[0]))
#print(m)
#Iterating through the number of images we need to print
for i in range(hidden_size):
    grid[i].imshow(params['Wlayer1'][:,i].reshape((m,-1)))

plt.show()
```

Code for initialized weights

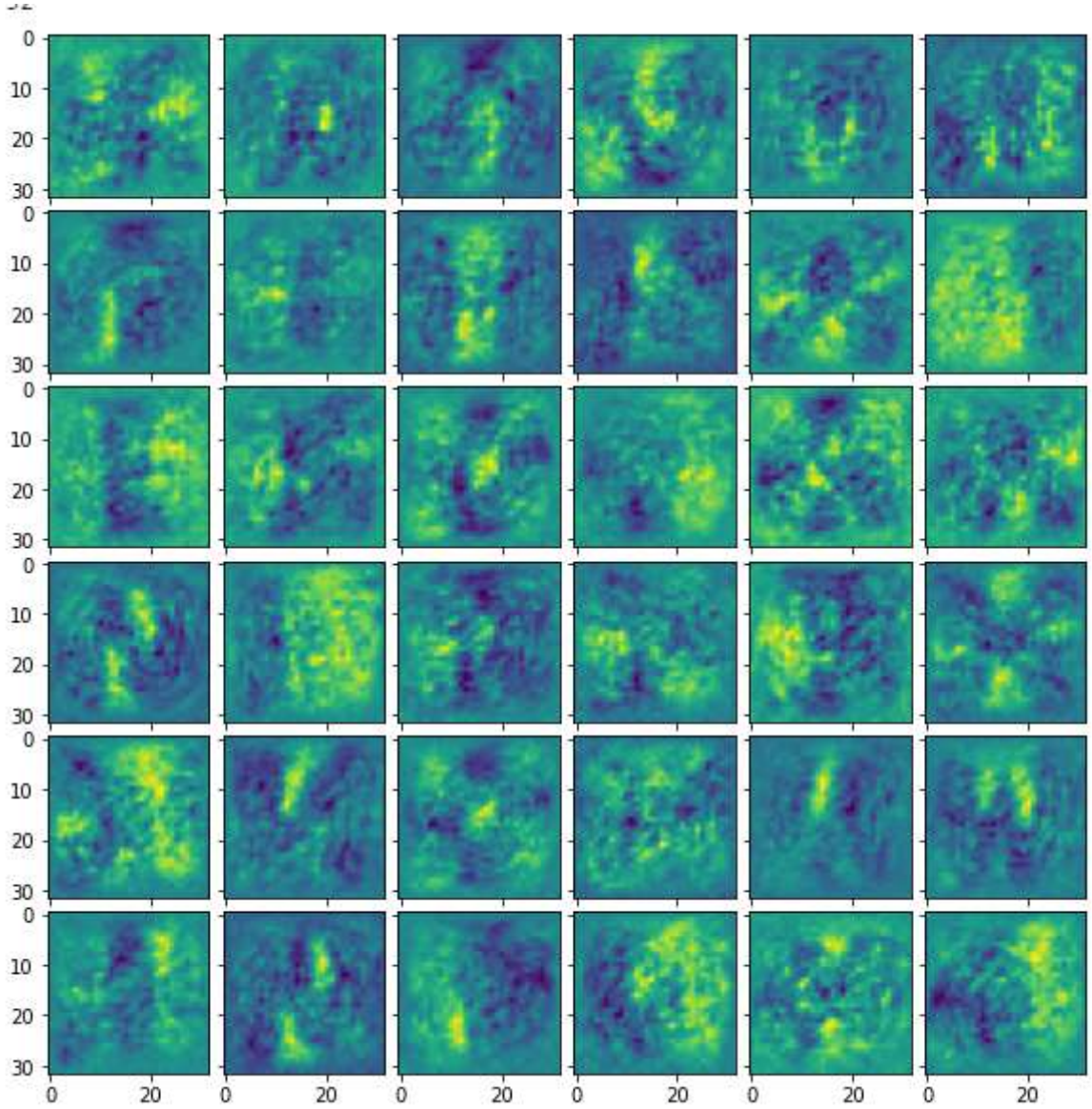
```
# Creating a new dictionary to avoid change in the original one
paramscopy = {}

#initializing weights
initialize_weights(train_x.shape[1],hidden_size,paramscopy,'layer1')
initialize_weights(hidden_size,train_y.shape[1],paramscopy,'output')

fig = plt.figure(figsize=(10, 10))

grid = ImageGrid(fig, 111,
                 nrows_ncols=(8, 8),
                 axes_pad=0.1,
                 )

# Spliting the dimension equally to reshape
m = int(np.sqrt(paramscopy['Wlayer1'].shape[0]))
#print(m)

for i in range(hidden_size): #Iterating through the number of images we need to print
    grid[i].imshow(paramscopy['Wlayer1'][:,i].reshape((m,-1)))

plt.show()
```

Q3.4 (3 points Code+WriteUp)

Images in the validation set for which the y2 values is maximum



Visualization of final layer weights



Code for the printing the maximum y2 value

```
# Visualizing the The images in the test set
id=[]
for j in range(valid_y.shape[1]): # 36 class
    for i in range(valid_y.shape[0]): # no of datasets
        if  valid_y[i,j] == 1:
            id.append(i)
            break # Collecting the index only onces
#print(id)
#id - index of position of each individual classes(36) in test set collected onces
# Creating the test images with 36 classes
testim = valid_x[id,:]

fig = plt.figure(figsize=(10, 10))
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(6, 6),  # creates 2x2 grid of axes
                 axes_pad=0.1,  # pad between axes in inch.
                 )
m = int(np.sqrt(testim.shape[1]))
#print(m)
for i in range(testim.shape[0]): #Iterating through the number of images we need to print
    grid[i].imshow((testim[i, :].reshape((m,-1)).T))

plt.show()
```

Code for visualization of final layer weights

```
#Creating a dictionary to avoid damaging the old one
paramscopy1 = {}
initialize_weights(train_x.shape[1],hidden_size,paramscopy1,'layer1')
initialize_weights(hidden_size,train_y.shape[1],paramscopy1,'output')
post_act1 =forward(train_x, paramscopy1, 'layer1',sigmoid)
post_act2 = forward(post_act1,paramscopy1,'output',softmax)
```

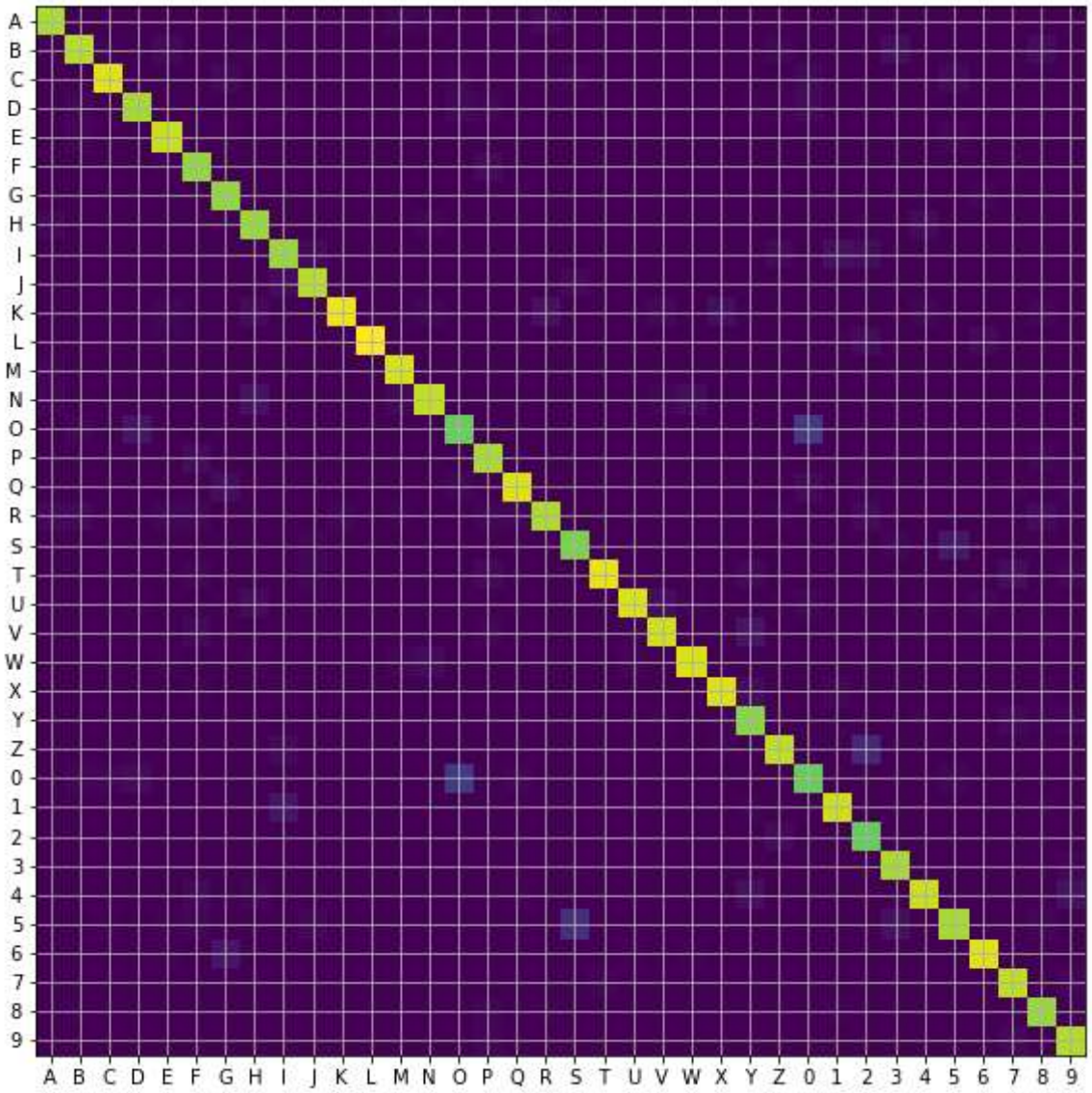
```
res =np.empty ((train_y.shape[1],params['Wlayer1'].shape[0]))
idx=[]
#Checking for one data each which corresponds to one of the 36 class. This is essentially a identity matriz (36)
for j in range(train_y.shape[1]): # 36 class
    for i in range(train_y.shape[0]): #10800 Data
        if  train_y[i,j] == 1:
            idx.append(i)
            break

            # Performing the multiplication with the output
res = train_y[idx,:]@ (params['Woutput']).T @ (params['Wlayer1']).T
```

```
fig = plt.figure(figsize=(10, 10))
grid = ImageGrid(fig, 111,
                 nrows_ncols=(6, 6),
                 axes_pad=0.1,
                 )
m = int(np.sqrt(res.shape[1]))
#print(m)

for i in range(res.shape[0]): #Iterating through the number of images we need to print
    grid[i].imshow(res[i, :].reshape((m,-1)))

plt.show()
```

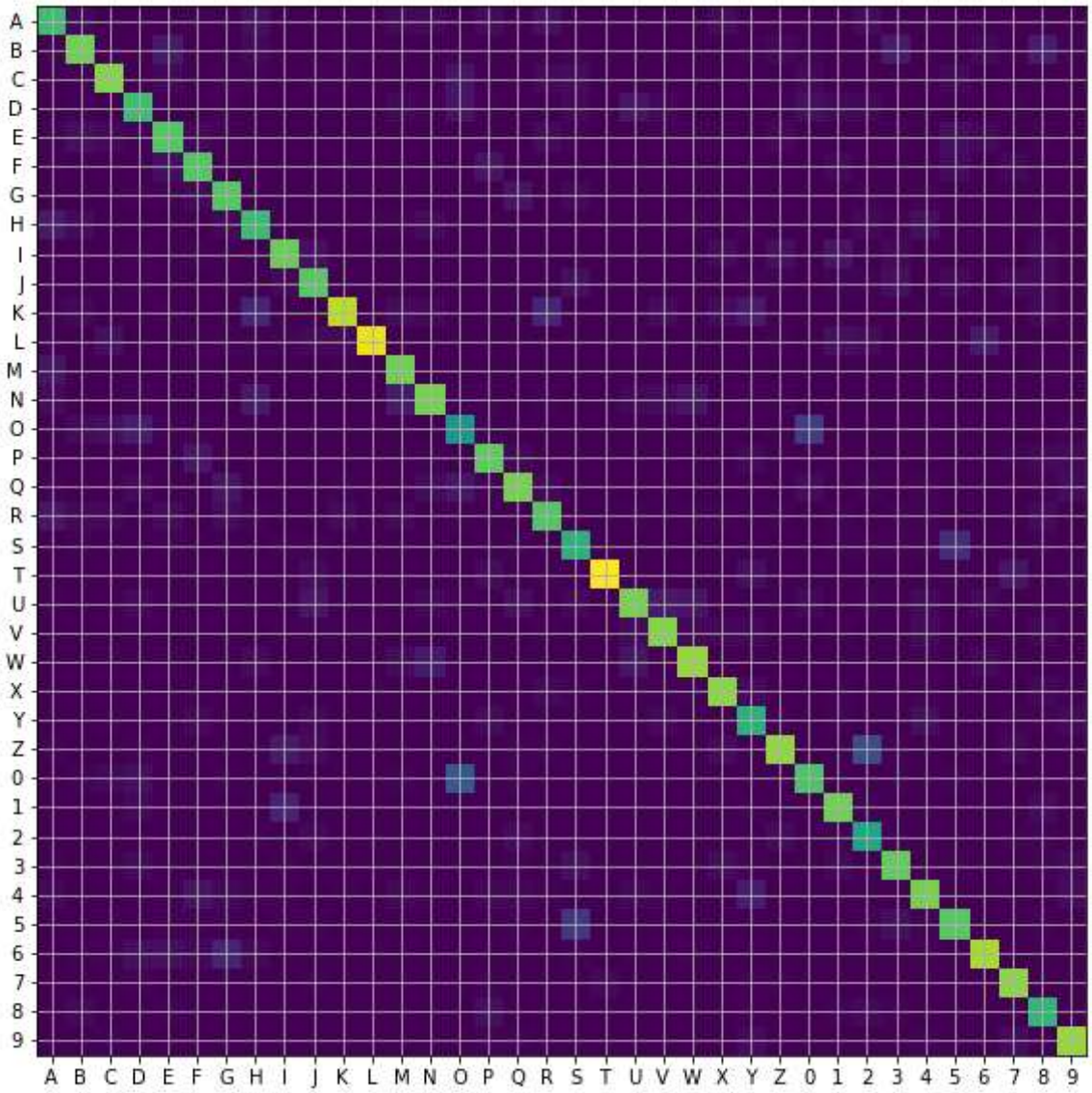
Q3.5 (4 points Code+WriteUp)

From the confusion matrix of the testset and trainset.The brighter the cell is =, the larger the number it was in that cell. Since, the diagonal is heavly brightened, we can conclude that the accuracy is good. Nevertheless, there are some mispairs like 0 and O , S and 5 , Z and 2, 6 and G. But these mispairs are more common features, which may be the reason for mismatch and they are often mismatched by human eye as well.

Confusion Matrix for Trainset



Confusion Matrix for Testset



Code for Confusion matrix:

```
#Finding th prediciton using train datset
post_act1_train =forward(train_x, params, 'layer1',sigmoid)
post_act2_train = forward(post_act1_train,params,'output',softmax)
loss , accuracy = compute_loss_and_acc(train_y, post_act2_train)
print(post_act2_train.shape); print(accuracy)
```

```
(10800, 36)
0.8675
```

```
valid_x, valid_y = valid_data['valid_data'], valid_data['valid_labels']
```

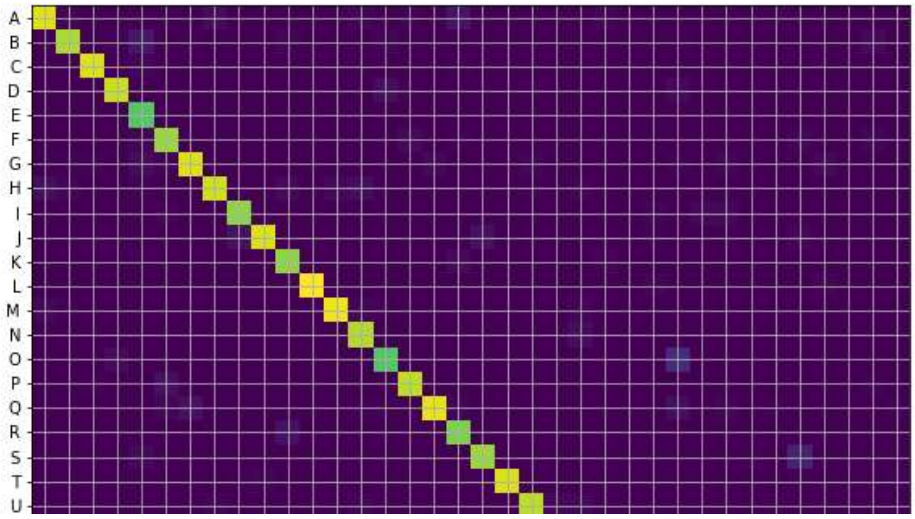
```
#Finding the prediction usinf test dataset
post_act1_valid =forward(valid_x, params, 'layer1',sigmoid)
post_act2_valid = forward(post_act1_valid,params,'output',softmax)
loss , accuracy = compute_loss_and_acc(valid_y, post_act2_valid)
print(post_act2_valid.shape); print(accuracy)
```

```
(3600, 36)
0.7508333333333334
```

```
import matplotlib.pyplot as plt
confusion_matrix_train = np.zeros((train_y.shape[1],train_y.shape[1]))
confusion_matrix_valid = np.zeros((valid_y.shape[1],valid_y.shape[1]))

 #for train conf matrix
for i in range(train_y.shape[0]):
    #Calculating the max probable position of the class
    x_train = np.argmax(post_act2_train[i, :])
    # Checking the position of 1 in hot vector
    y_train = np.where(train_y[i, :]==1)
    confusion_matrix_train[x_train, y_train] += 1
print(np.trace(confusion_matrix_train)/np.sum(confusion_matrix_train))
for i in range(valid_y.shape[0]):
    x_valid = np.argmax(post_act2_valid[i, :])
    y_valid = np.where(valid_y[i, :]==1)
    confusion_matrix_valid[x_valid, y_valid] += 1
```

```
0.8624074074074074
```

```
#Plotting the confusion matrix
import string
fig = plt.figure(figsize=(10, 10))
plt.imshow(confusion_matrix_train,interpolation='nearest')
plt.grid(True)
plt.xticks(np.arange(36),string.ascii_uppercase[:26] + ''.join([str(_) for _ in range(10)]))
plt.yticks(np.arange(36),string.ascii_uppercase[:26] + ''.join([str(_) for _ in range(10)]))
plt.show()
```



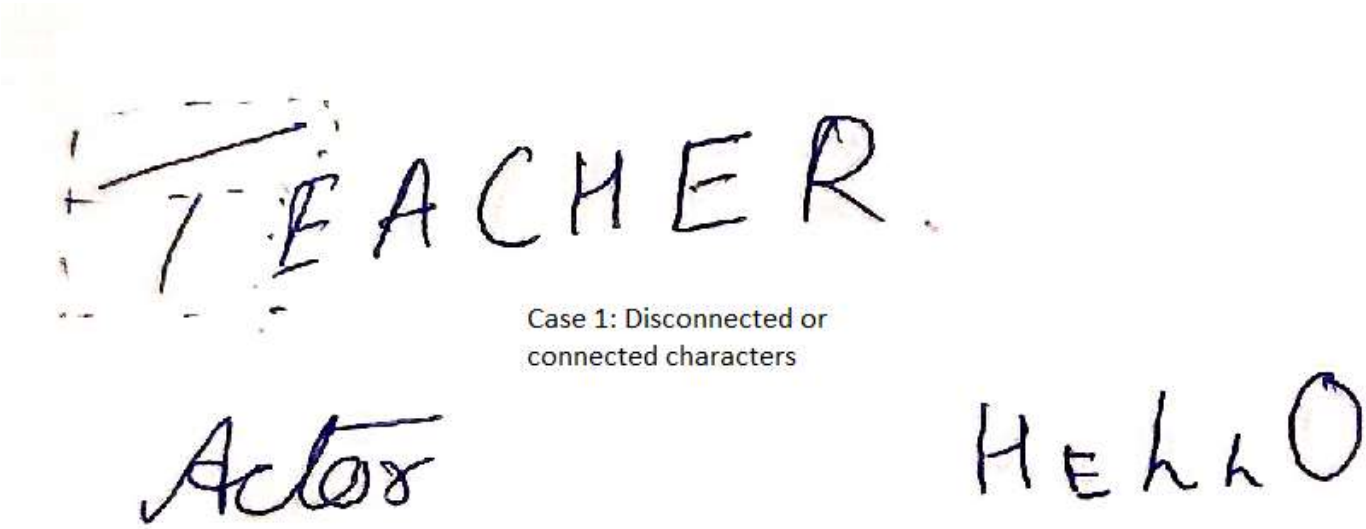
```
import string
fig = plt.figure(figsize=(10, 10))
plt.imshow(confusion_matrix_valid,interpolation='nearest')
plt.grid(True)
plt.xticks(np.arange(36),string.ascii_uppercase[:26] + ''.join([str(_) for _ in range(10)]))
plt.yticks(np.arange(36),string.ascii_uppercase[:26] + ''.join([str(_) for _ in range(10)]))
plt.show()
```

Q4 Extract Text from Images (35 points)

Q4.1 (3 points WriteUp)

The two big assumption we make are :

1. Each character is continous and fully connected. Every character is isolated from another character. This method works on the connections of the character. So for words like shown below or the one with cursive writing wont be detected properly.
2. The characters are of similar size. If few of the character are smaller, then it might get filtered out and wont be detected properly. One example is show below.



Case 2 : Character with different sizes

Q4.2 (13 points Code+WriteUp)

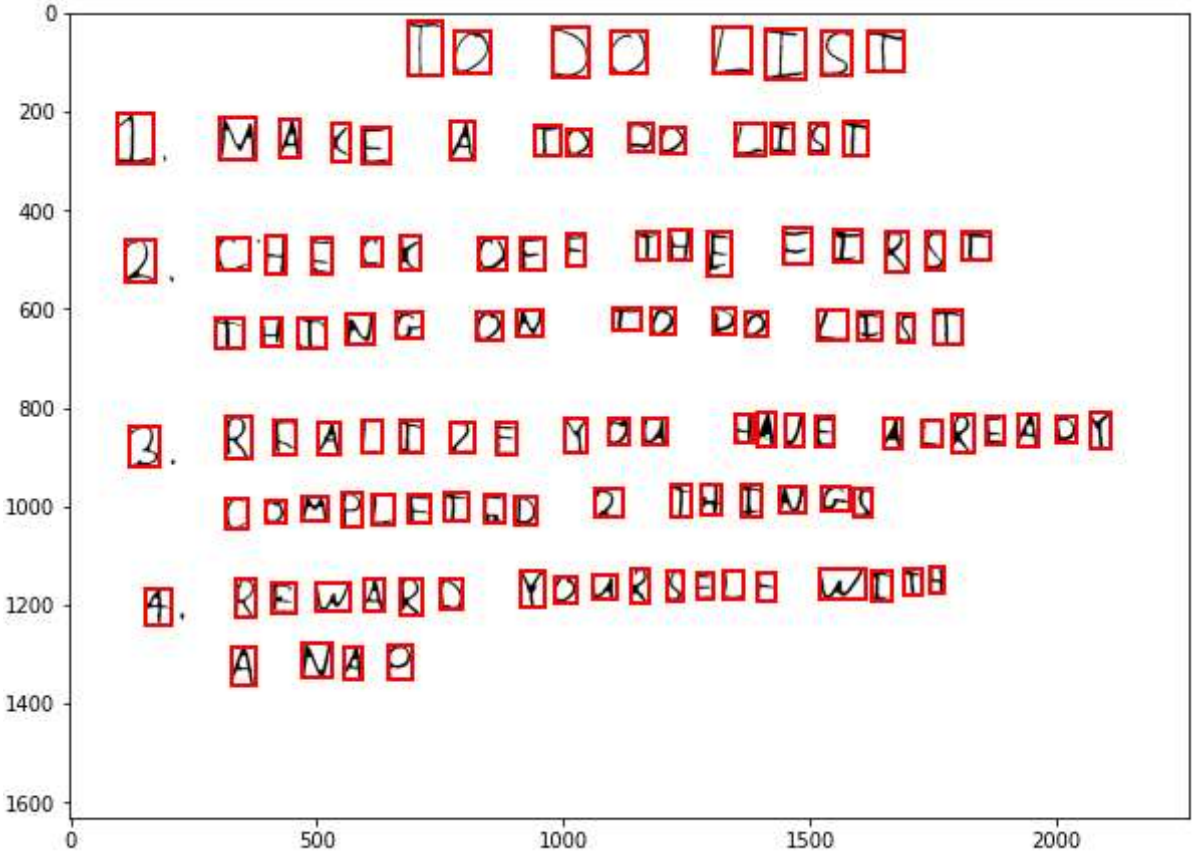
Code:

```
import numpy as np
import skimage
import skimage.measure
import skimage.color as sc
import skimage.restoration as sr
import skimage.io
import skimage.filters as sf
import skimage.morphology
from skimage.morphology import closing, square
import skimage.segmentation


def findLetters(image):
    bboxes = []

    bw = None
    # Calculating the noise in the image
    sig_est = sr.estimate_sigma(image, multichannel =True, average_sigmas=True)
    #print(sig_est)

    #Denosing the image
    de_image = sr.denoise_bilateral(image,sigma_color=sig_est ,multichannel= True) #Denoising

    #Converting to Greyscale and float
    gray = sc.rgb2gray(de_image)
    gray = gray.astype('float')

    thresh = sf.threshold_otsu(gray) #taking threshold

    #Performing morphology
    bw= closing(gray < thresh, square(8))

    #Removing artifacts in the border
    cleared = skimage.segmentation.clear_border(bw)

    # label image regions
    label_image = skimage.measure.label(cleared,connectivity=2) #background 1 as white is the req background

    #Selecting the bounding boxes based on the mean are of the bounding boxes
    #Calculating the mean are
    meanarea = 0
    i=0
    for region in skimage.measure.regionprops(label_image):
        meanarea += region.area
        i +=1
    meanarea = meanarea/i

    #Looping throught each boundung boxes and finding the optimal ones
```

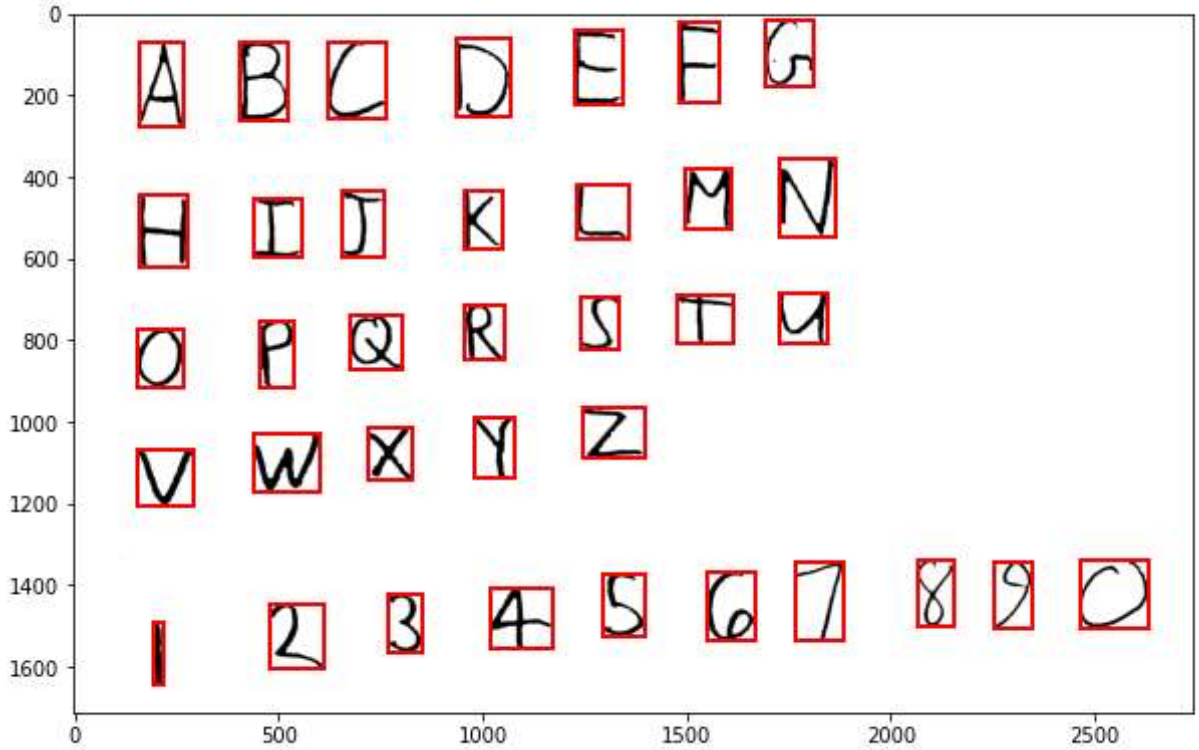
```
    #Looping throught each boundung boxes and finding the optimal ones
    for region in skimage.measure.regionprops(label_image):
        if region.area >=meanarea/3:
        # draw rectangle around segmented image
            minr, minc, maxr, maxc = region.bbox
            bboxes.append([minr, minc, maxr, maxc])

   #inverting the image as the required output should be in yellow background (white when cmap = gray)
    bw = (1-bw)
    return bboxes, bw
```

Q4.3 (6 points WriteUp)

Images with bounding boxes

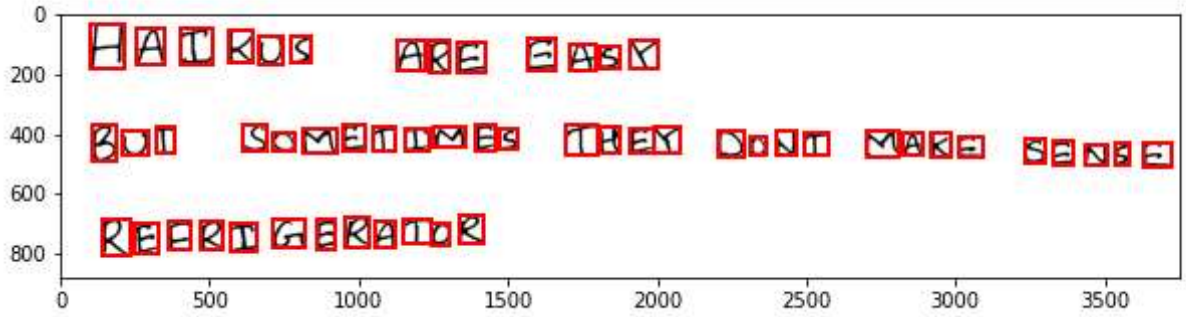

TO DO LIST
1. MAKE A TO DO LIST
2. CHECK OFF THE ERRO
THING ON TO DO LIST
3. REALIZE YOU HAVE ACHEAD
COMPLETED 2 THINGS
4. REWARD YOURSELF WITH
A NAP

115

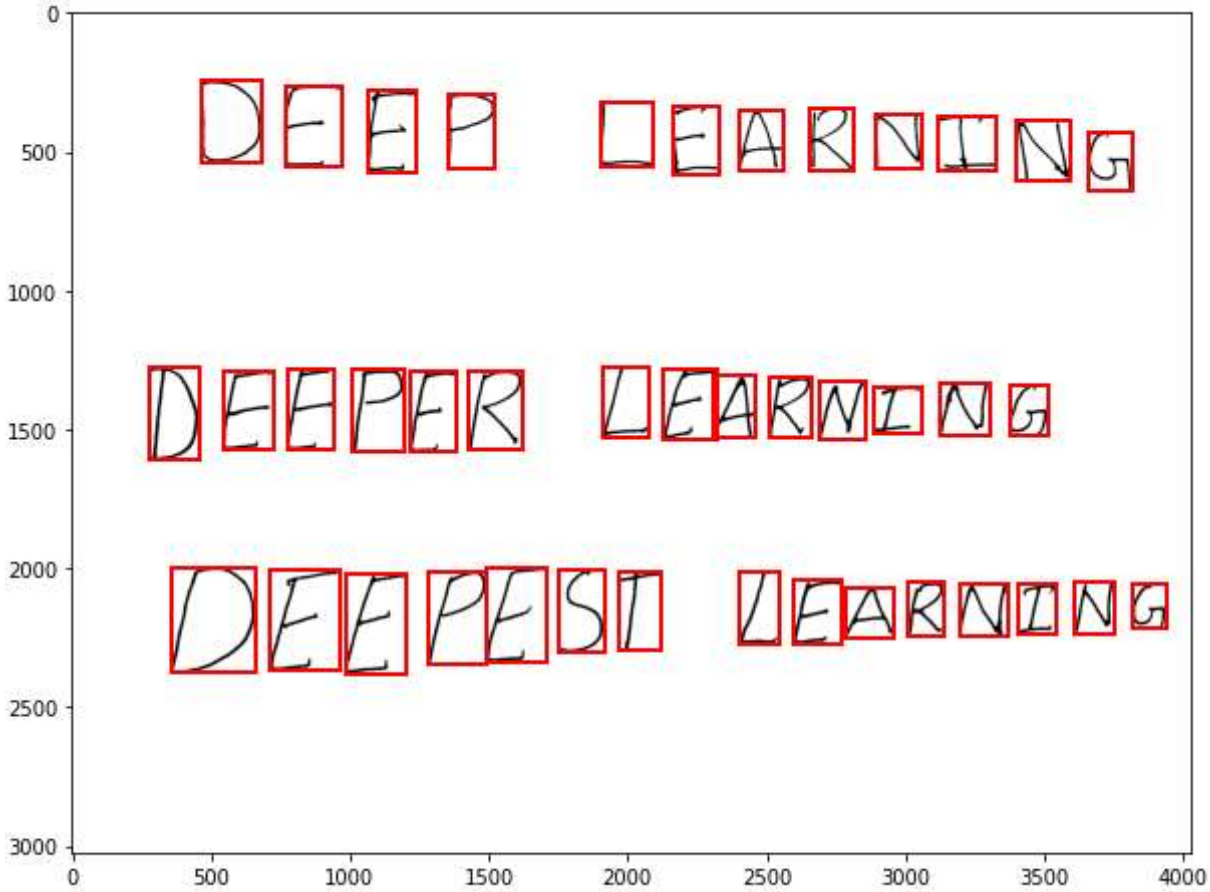

A B C D E F G
H I J K L M N
O P Q R S T U
V W X Y Z
1 2 3 4 5 6 7 8 9 0

36

36


HAIKUS ARE EASY
BUT SOMETIMES THEY DONT MAKE SENSE
REFRIGERATOR

54


DEEP LEARNING
DEEPER LEARNING
DEEPEST LEARNING

Code:

```
def sortedbb(img, thres):

    sorted = []

    #Reading the image
    im1 = skimage.img_as_float(skimage.io.imread(os.path.join('images',img)))

    # Extracting the bounding boxes
    bboxes, bw = findLetters(im1)
    plt.imshow(bw, cmap='gray')

    #Initialization for centers and height and width
    # For centers
    ybb = []
    xbb = []

    h = []
    w= []

    #Looping through bounding boxes
    for bbox in bboxes:
        minr, minc, maxr, maxc = bbox
        rect = matplotlib.patches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                 fill=False, edgecolor='red', linewidth=2)
        plt.gca().add_patch(rect)

        #Centers
        ybb.append((bbox[2]+bbox[0])/2)
        xbb.append((bbox[1]+bbox[3])/2)
        #Height and width
        h.append(bbox[2]-bbox[0])
        w.append(bbox[3]-bbox[1])

    plt.show()

    #Taking the mean height and width in the boxes
    hm = np.mean(np.asarray(h))
    wm = np.mean(np.asarray(w))

    #print(hm//20, wm//20)

    #Creating a array with y coordinate, x coordinate of center and the initial arrangement
    ybb = np.asarray(ybb).reshape((-1,1))
    xbb = np.asarray(xbb).reshape((-1,1))
    id = np.asarray(np.arange(ybb.shape[0])).reshape((-1,1))
    #print(id)

    centerbb = np.hstack((ybb, xbb, id))
```

```
    #print(id)

    centerbb = np.hstack((ybb, xbb, id))

    #Sorting the center wrt to y axis
    idx = np.argsort(centerbb[:,0])

    ncenbb = np.copy(centerbb[idx,:])
    #print(ncenbb[:, 2].astype(int))

    #Sorting algorithm
    sort = []# Provides the indicies of boxes after sorting
    i =0
    print(ybb.shape[0])
    while(i < ybb.shape[0]):

        tmp = []

        if i==0:
            tmp.append(ncenbb[i,2])
            #print(ncenbb[i,2].astype(int))
            i +=1
        j =0

        #Clustering the rows based on a threshold
        while(((ncenbb[i+j,0] - ncenbb[i,0] ) < thres) ):
            tmp.append(ncenbb[i+j,2])
            #print(i+j)
            #print(ncenbb[i+j,2].astype(int))
            j+= 1
            if (i+j >= ybb.shape[0]):
                break

        #print(tmp)
        tid = np.asarray(tmp).astype(int)
        #print(tid)

        #Sorting the Clustered row wrt to xaxis
        tidx =np.argsort(centerbb[tid,1]).astype(int)
        #print(tid[tidx])

        #Stacking the cluested row into a list
        sort = np.hstack((sort, tid[tidx]))

        #Striding to the next row
        i = i + j
    #print(sort)
```

```
        #Striding to the next row
        i = i + j
    #print(sort)
    #Cropping the bboxes and processsing it
    for i in sort:

        nb = np.asarray(bboxes)
        #print(nb[int(i),0])
        minr, minc, maxr, maxc =nb[int(i),:]

        h = int((maxr - minr))
        w = int((maxc - minc))
        h_pad, w_pad = 0, 0

        #For each images the padding value is different so padding based on the height and width value
        if h > w:
            h_pad = h//20
            w_pad = (h-w)//2+h_pad
        elif h < w:
            w_pad = w//20
            h_pad = (w-h)//2+w_pad

        # For square padding
        if h_pad> w_pad:
            pad = h_pad
        elif h_pad < w_pad:
            pad = w_pad

        #Aspect ratio Not required for this code
        r = w/h
        #print(h_pad, w_pad)

        crop = bw[minr:maxr, minc:maxc]

        crop = np.pad(crop, ((pad,pad), (pad,pad)), 'constant', constant_values=1.0)

        # For Refernce (Tuning the parameters)
        # if (img == '01_list.jpg'):
        #     crop = np.pad(crop, ((20, 20), (20, 20)), 'constant', constant_values=1.0)
        # else:
        #     crop = np.pad(crop, ((int(r*50), int(r*50)), (50, 50)), 'constant', constant_values=1.0)
        #*************************************
        crop =crop*255 #Needed before resizing
        crop_size = skimage.transform.resize(crop,(32,32), anti_aliasing=True)

        #resizing affected the values in the image, so nomalizing back to 0,1
        crop_size = (crop_size- np.min(crop_size))/(np.max(crop_size)- np.min(crop_size))
```

```
        crop =crop*255 #Needed before resizing
        crop_size = skimage.transform.resize(crop,(32,32), anti_aliasing=True)

        #resizing affected the values in the image, so nomalizing back to 0,1
        crop_size = (crop_size- np.min(crop_size))/(np.max(crop_size)- np.min(crop_size))
        #Intensifing the pixel values in the center
        crop_size = skimage.morphology.erosion(crop_size,  np.array([[0, 1, 0], [1, 1, 1], [0, 1, 0]]))


        # plt.imshow(crop_size)
        # plt.show()
        sorted.append(crop_size.T)


    return sorted
```

```
import  nbimporter
import os
import matplotlib.pyplot as plt
import matplotlib.patches
import q2
from q2 import *

# do not include any more libraries here!
# no opencv, no sklearn, etc!
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

#Creating list of sorted bboxes for each images
sortedim1 =[]
sortedim2 = []
sortedim3 = []
sortedim4 =[]

for img in os.listdir('images'):
    if (img == '01_list.jpg'):
        sortedim1 = sortedbb(img, thres = 150)
    if (img == '02_letters.jpg'):
        sortedim2 = sortedbb(img, thres = 200)
    if (img == '03_haiku.jpg'):
        sortedim3 = sortedbb(img, thres = 200)
    if (img == '04_deep.jpg'):
        sortedim4 = sortedbb(img, thres = 200)
```

```
# Saving the bboxes in the pickle for q7
import pickle
# print(np.asarray(sortedim1).shape)
# print(np.asarray(sortedim2).shape)
# print(np.asarray(sortedim3).shape)
# print(np.asarray(sortedim4).shape)
with open('q4img1.pickle', 'wb') as handle:
    pickle.dump(sortedim1, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('q4img2.pickle', 'wb') as handle:
    pickle.dump(sortedim2, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('q4img3.pickle', 'wb') as handle:
    pickle.dump(sortedim3, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('q4img4.pickle', 'wb') as handle:
    pickle.dump(sortedim4, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Q4.4 (13 points Code+WriteUp)

Results from the extracted image and the original answers. From the results we get about 80%. The results are good except for the To Do list.

```
Correct Ans1  : T O D O L I S T 1 M A K E A T O D O L I S T 1 C H E C K O F F T H E F I R S T T H I N G O N T O D O L I S T 3 R E A L I Z E Y O U H A V E A L R E A D Y C O M P L E T E D 2 T H I N G S 4 R
E W A R D Y O U R S E L F W I T H A N A P
Obtained Ans1:  T Q D Q L I S T I N A K E A T O D U L I S T Z L H K G K D F F T T H E F I R S T T M I N G Q N O D Q L I S T 3 R E A L I Z E Y O U Y H H I A N U E G A L R 6 A D X G O M P L E T G D I S 4 R
E W A R D Y O U R S E L F W I T H A N A P

Correct Ans2  : 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 1 2 3 4 5 6 7 8 9 0
Obtained Ans2:  A B C D E F G H I J K L M N Q P Q K S T U V W X Y Z X Z 3 G S G 7 X 9 D

Correct Ans3  : H A I K U S A R E E A S Y B U T S O M E T I M E S T H E Y D O N T M A K E S E N S E R E F R I G E R A T O R
Obtained Ans3:  H A I W U S A R G H A G X B U T S Q M E T I M E S T R E X D D W T M A K 6 S E N G E R B G R I G E R R T Q R

Correct Ans4  : D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G
Obtained Ans4:  D E E P L E A R Z I K G D E E P E R L E A R N I K G D E E P E S T L E A R N I N G
```

Code:

Q4.4 (13 points Code+WriteUp)

Now you will load the image, find the character locations, classify each one with the network you trained in **Q3.1**, and return the text contained in the image. Be sure you try to make your detected images look like the images from the training set. Visualize them and act accordingly.

Please include the write up answer and the screenshot of code to theory.ipynb

```
#Trainging in the NN and classification
def getresult(x):
    st = ' '
    for i in range(x.shape[0]):
        img = x[i,:,:].reshape((-1,32*32)) #Reshaping according to the input need in NN
        #print(img.shape)

        postact1 = forward(img, params, 'layer1')
        postact2 = forward(postact1, params, 'output', softmax)
        #print(postact2.shape)

        idx = np.argmax(postact2[0, :])
        #print(idx)
        st = st + letters[idx] + ' '
    return st
```

```
# load the weights
# run the crops through your neural network and print them out
import pickle
import string
import numpy as np
x1 = np.array(sortedim1)
x2 = np.array(sortedim2)
x3 = np.array(sortedim3)
x4 = np.array(sortedim4)
letters = np.array([_ for _ in string.ascii_uppercase[:26]] + [str(_) for _ in range(10)])

#Importing parameters from q3
params = pickle.load(open('q3_weights.pickle','rb'))

Ans1 = 'T O D O L I S T 1 M A K E A T O D O L I S T 1 C H E C K O F F T H E F I R S T T H I N G O N T O D O L I S T 3 R E A L I Z E Y O U H A V E A L R E A D Y C O M P L E T E D 2 T H I N G S 4 R E W A R D
Ans2 = '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 1 2 3 4 5 6 7 8 9 0'
Ans3 = 'H A I K U S A R E E A S Y B U T S O M E T I M E S T H E Y D O N T M A K E S E N S E R E F R I G E R A T O R'
Ans4 = 'D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G'
st1 = getresult(x1)
```

```
#Importing parameters from q3
params = pickle.load(open('q3_weights.pickle','rb'))

Ans1 = 'T O D O L I S T 1 M A K E A T O D O L I S T 1 C H E C K O F F T H E F I R S T T H I N G O N T O D O L I S T 3 R E A L I Z E Y O U H A V E A L R E A D Y C O M P L E T E D 2 T H I N G S 4 R E W A R
Ans2 = '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 1 2 3 4 5 6 7 8 9 0'
Ans3 = 'H A I K U S A R E E A S Y B U T S O M E T I M E S T H E Y D O N T M A K E S E N S E R E F R I G E R A T O R'
Ans4 = 'D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G'
st1 = getresult(x1)
st2 = getresult(x2)
st3 = getresult(x3)
st4 = getresult(x4)


# print(len(st1))
# print(len(st2))
# print(len(st3))
# print(len(st4))

print('\nCorrect Ans1  :', Ans1)
print('Obtained Ans1:', st1)
# print('\n Accuracy = ', cor1/len(st1)*100)

print('\nCorrect Ans2  :', Ans2)
print('Obtained Ans2:', st2)

print('\nCorrect Ans3  :', Ans3)
print('Obtained Ans3:', st3)

print('\nCorrect Ans4  :', Ans4)
print('Obtained Ans4:', st4)
# YOUR CODE HERE
#raise NotImplementedError()
```

Q5 Image Compression with Autoencoders [Extra Credit](25 points)

Q5.1.1 [Extra Credit](10 points Code)

YOUR ANSWER HERE

Q5.2 [Extra Credit](3 points Code+WriteUp)

YOUR ANSWER HERE

Q5.3.1 [Extra Credit](4 points Code+WriteUp)

YOUR ANSWER HERE

Q5.3.2 [Extra Credit](3 points Code+WriteUp)

YOUR ANSWER HERE

Q6 Comparing against PCA [Extra Credit](15 Points)

Q6.1 [Extra Credit](4 points Code+WriteUp)

YOUR ANSWER HERE

Q6.2 [Extra Credit](4 points Code+WriteUp)

YOUR ANSWER HERE

Q6.3 [Extra Credit](4 points Code+WriteUp)

YOUR ANSWER HERE

Q6.4 [Extra Credit](3 points Code+WriteUp)

YOUR ANSWER HERE

Q7 PyTorch (40 points)

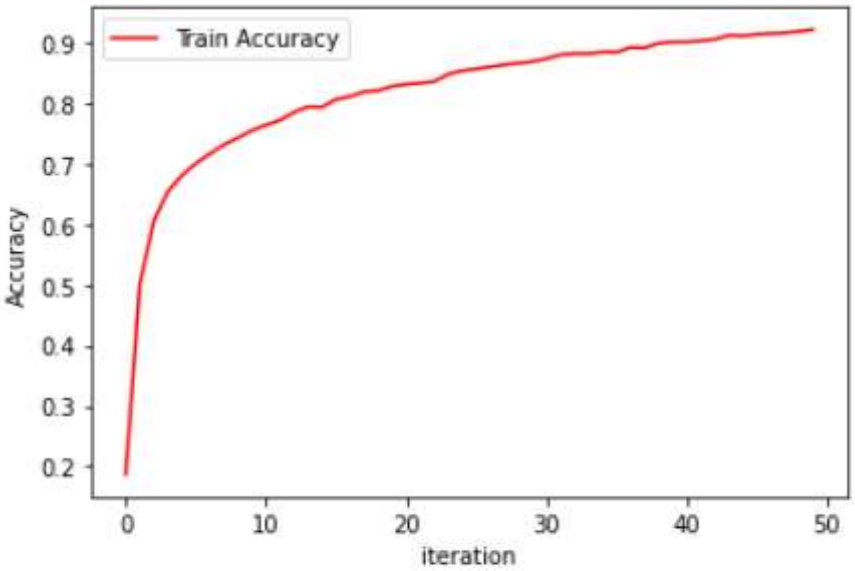
Q7.1.1 (10 points Code+WriteUp)

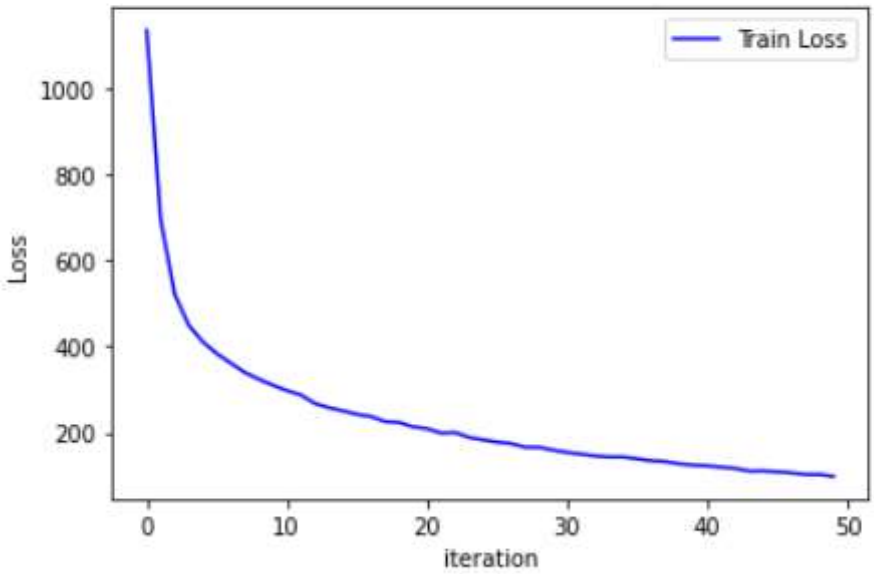
The final validation accuracy = 0.78

Results:

```
 Validation set: itr: 32         loss: 89.64     avg_acc : 0.78
itr: 34          loss: 143.41    acc : 318.63    avg_acc : 0.89

 Validation set: itr: 34         loss: 88.81     avg_acc : 0.79
itr: 36          loss: 134.52    acc : 321.33    avg_acc : 0.89

 Validation set: itr: 36         loss: 86.97     avg_acc : 0.79
itr: 38          loss: 127.30    acc : 323.90    avg_acc : 0.90

 Validation set: itr: 38         loss: 91.79     avg_acc : 0.78
itr: 40          loss: 122.63    acc : 324.70    avg_acc : 0.90

 Validation set: itr: 40         loss: 88.41     avg_acc : 0.79
itr: 42          loss: 116.65    acc : 326.17    avg_acc : 0.91

 Validation set: itr: 42         loss: 90.57     avg_acc : 0.78
itr: 44          loss: 111.02    acc : 328.17    avg_acc : 0.91

 Validation set: itr: 44         loss: 90.01     avg_acc : 0.79
itr: 46          loss: 106.31    acc : 329.57    avg_acc : 0.92

 Validation set: itr: 46         loss: 89.41     avg_acc : 0.79
itr: 48          loss: 102.50    acc : 330.90    avg_acc : 0.92

 Validation set: itr: 48         loss: 90.99     avg_acc : 0.78
```

Plots:


Train Accuracy
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50
iteration


Train Loss
Loss
1000
800
600
400
200
0
10
20
30
40
50
iteration

Code:

```
import nbimporter
import numpy as np
import scipy.io
from torch import nn, optim
import torch.nn.functional as F
from q2 import *
import torch
import torchvision.datasets
import matplotlib.pyplot as plt
from torch.utils.data import TensorDataset, DataLoader
from torch.autograd import Variable
max_iters = 50
batch_size = 30
learning_rate = 2.55e-2
hidden_size = 64
```

```
#Loading the dataset
train_data = scipy.io.loadmat('data/nist36_train.mat')
valid_data = scipy.io.loadmat('data/nist36_valid.mat')

train_x, train_y = train_data['train_data'], train_data['train_labels']
valid_x, valid_y = valid_data['valid_data'], valid_data['valid_labels']

#Converting to Tensor
tensor_x = torch.Tensor(train_x).float() # transform to torch tensor
tensor_y = torch.Tensor(train_y)

val_x, val_y = torch.Tensor(valid_data['valid_data']).float(), torch.Tensor(valid_data['valid_labels'])

#Loading to dataloader
trainset_loader = DataLoader(TensorDataset(tensor_x, tensor_y), batch_size= batch_size, shuffle=True, num_workers=1)
validset_loader = DataLoader(TensorDataset(val_x, val_y), batch_size= batch_size, shuffle=True, num_workers=1)


#Creating a Class of the torch module:
D = train_x.shape[1]
C = train_y.shape[1]
#print(C)
tot = tensor_y.shape[1]
#print(tot)

class Net(nn.Module):

    def __init__(self, D, hidden_size, C ):
        super(Net, self).__init__()
```

```
class Net(nn.Module):

    def __init__(self, D, hidden_size, C ):
        super(Net, self).__init__()

        self.fc1 = nn.Linear(D, hidden_size) #Two layers input Dimension is number of datapoints i, middle layer
        #is the hidden layer
        self.fc2 = nn.Linear(hidden_size, C)#output layer is the number of classes

    def forward(self, x):

        x = torch.sigmoid( self.fc1(x))
        # If the size is a square, you can specify with a single number
        x = self.fc2(x)
        return x

# use_cuda = torch.cuda.is_available()
#device = torch.device("cuda" if use_cuda else "cpu")
model = Net(D, hidden_size, C )

#Gradient Descent with momentum
optimizer = optim.SGD(model.parameters(), lr=learning_rate , momentum=0.9)

train_acclist =[]

train_losslist =[]


# with default settings, you should get loss < 150 and accuracy > 80%
for itr in range(max_iters):
    total_loss = 0

    total_acc =0
    for batch_idx, (data, target) in enumerate(trainset_loader):
        data, target = Variable(data), Variable(target)
        t_x, t_y =data, target

        #Cross entropy doesnt take hot vectors directly
        t_y = torch.max(t_y, 1)[1]

        #print(t_y.shape)
        #print(t_y)
        prediction = model(t_x)


        #F.nll_loss not working
        #Calculating loss
        loss = F.cross_entropy(prediction, t_y)
```

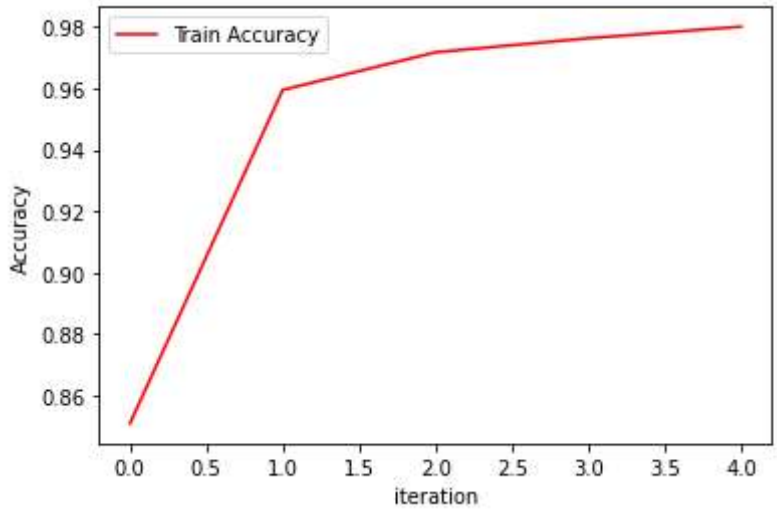
```
        #F.nll_loss not working
        #Calculating loss
        loss = F.cross_entropy(prediction, t_y)
        total_loss += loss.item()


        _, preds = torch.max(prediction, dim=1)
        accuracy = torch.tensor(torch.sum(preds == t_y).item() / len(preds))
        total_acc += accuracy

        #backward
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
    avg_acc = total_acc/(batch_idx+1)
    if itr % 2 == 0:

        print("itr: {:02d} \t loss: {:.2f} \t acc : {:.2f} \t avg_acc : {:.2f}".format(itr,total_loss,total_acc, avg_acc))


    model.eval()  # set evaluation mode
    valid_loss = 0
    valid_acc = 0
    with torch.no_grad():
        for v_idx, (data, target) in enumerate(validset_loader):

            v_x, v_y = data, target

            v_pred = model(v_x)

            v_y = torch.max(v_y, 1)[1]

            loss = F.cross_entropy(v_pred, v_y)

            valid_loss += loss
            # Vloss.append(valid_loss)

            _, vpreds = torch.max(v_pred, dim=1)
            accuracy = torch.tensor(torch.sum(vpreds == v_y).item() / len(vpreds))

            valid_acc += accuracy


    val_avg_acc= valid_acc/(v_idx+1)

    if itr % 2 == 0:
        print("\n Validation set: itr: {:02d} \t loss: {:.2f} \t avg_acc : {:.2f}".format(itr,valid_loss, val_avg_acc))
```

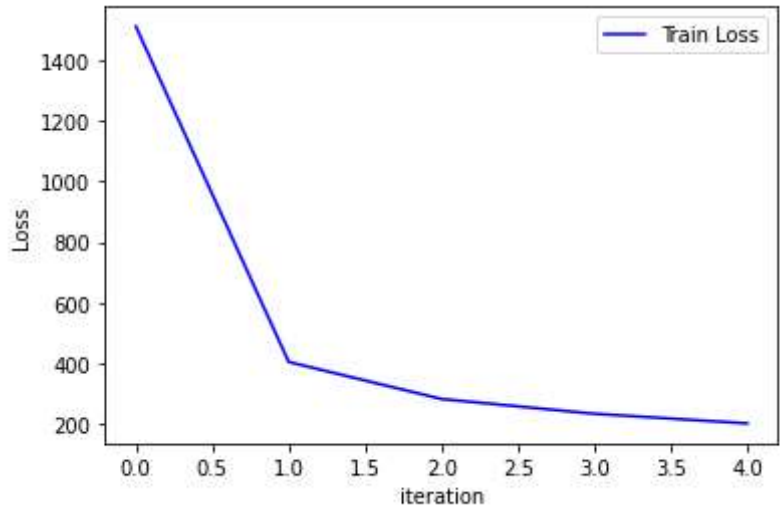
```
val_avg_acc= valid_acc/(v_idx+1)

if itr % 2 == 0:
    print("\n Validation set: itr: {:02d} \t loss: {:.2f} \t avg_acc : {:.2f}".format(itr,valid_loss, val_avg_acc))


train_acclist.append(avg_acc)
train_losslist.append(total_loss)
```

```
#Plotting the Graphs for accuracy and loss

a = np.arange(0, max_iters)
train_acclist =np.asarray(train_acclist)

train_losslist =np.asarray(train_losslist)



plt.figure(0)
plt.plot(a, train_acclist, 'r')
plt.legend(['Train Accuracy'])
plt.xlabel('iteration')
plt.ylabel('Accuracy')

plt.figure(1)
plt.plot(a, train_losslist, 'b')
plt.legend(['Train Loss', 'Validation Loss'])
plt.xlabel('iteration')
plt.ylabel('Loss')

plt.show()
```

Q7.1.2 (3 points Code+WriteUp)

Plots:





Code:

Q7.1.2 (3 Points Code+WriteUp)

Train a convolutional neural network with PyTorch on MNIST. Plot training accuracy and loss over time.

Please include the write up answer and the screenshot of code to theory.ipynb

```
import numpy as np
import scipy.io
import torch
import torchvision
from torch import nn, optim
import torch.nn.functional as F  # a lower level (compared to torch.nn) interface
from torch.utils.data import Dataset, DataLoader
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
from time import time

# Use GPU if available, otherwise stick with cpu
use_cuda = torch.cuda.is_available()
torch.manual_seed(123)

device = torch.device("cuda" if use_cuda else "cpu")
print("device = {}".format(device))
print("Get dataset")

mnist_train = MNIST(root="data", train=True, download=True, transform=transforms.ToTensor())
trainset_loader = DataLoader(mnist_train, batch_size=20, shuffle=True, num_workers=1)

mnist_test = MNIST(root="data", train=False, download=True, transform=transforms.ToTensor())
testset_loader = DataLoader(mnist_test, batch_size=20, shuffle=True, num_workers=1)

print("dataset size train, test")
print(trainset_loader.dataset.data.shape)
print(testset_loader.dataset.data.shape)

max_iters = 5
train_accmnist =[]
train_lossmnist =[]

#Creating a CNN with 2 layers of crosscorelation (conv2D) and 2 linear connection
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
```

```
#Creating a CNN with 2 layers of crosscorelation (conv2D) and 2 linear connection
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.lay1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=3,
                stride=1,
                padding=1,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.lay2 = nn.Sequential(
            nn.Conv2d(16, 32, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 10 classes
        #input dimension from the convolution based on the formula (w-k+2p)/s +1
        self.fc1 = nn.Linear(32*7*7, 196)
        self.fc2 = nn.Linear(196, 10)

    def forward(self, x):
        x = self.lay1(x)
        x = self.lay2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x= self.fc2(x)

        return x

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.001 , momentum=0.9)

for itr in range(max_iters):
    total_loss = 0

    total_acc =0
    for batch_idx, (data, target) in enumerate(trainset_loader):
        data, target = Variable(data), Variable(target)
        #print(batch_idx)
        t_x, t_y =data, target

        #Forward prop
```

```
for itr in range(max_iters):
    total_loss = 0

    total_acc =0
    for batch_idx, (data, target) in enumerate(trainset_loader):
        data, target = Variable(data), Variable(target)
        #print(batch_idx)
        t_x, t_y =data, target

        #Forward prop
        prediction = model(t_x)

        #F.nll_loss not working
        #Calculating Loss and accuracy
        loss = F.cross_entropy(prediction, t_y)
        total_loss += loss.item()


        _, preds = torch.max(prediction, dim=1)
        accuracy = torch.tensor(torch.sum(preds == t_y).item() / len(preds))
        total_acc += accuracy

        #backward
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    avg_acc = total_acc/(batch_idx+1)


    print("itr: {:02d} \t loss: {:.2f} \t acc : {:.2f} \t avg_acc : {:.2f}".format(itr,total_loss,total_acc, avg_acc))
    train_accmnist.append(avg_acc)
    train_lossmnist.append(total_loss)

    model.eval()  # set evaluation mode
    valid_loss = 0
    valid_acc = 0
    with torch.no_grad():
        for v_idx, (data, target) in enumerate(testset_loader):

            v_x, v_y = data, target

            v_pred = model(v_x)

            loss = F.cross_entropy(v_pred, v_y)
```

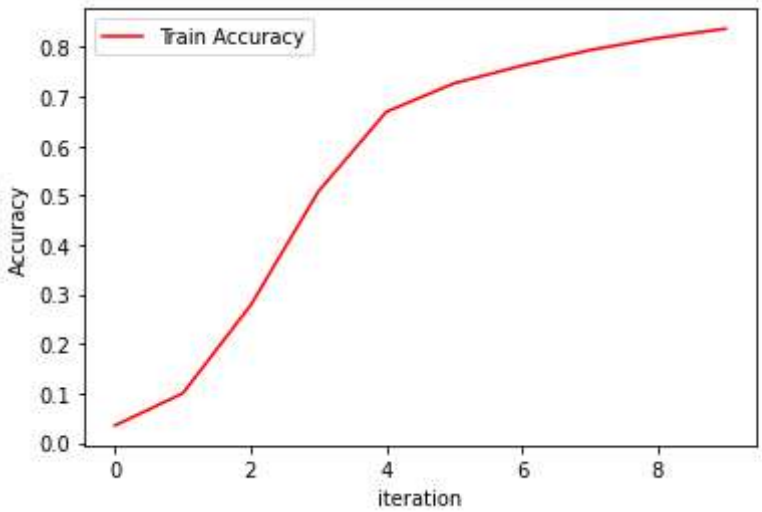
```
            loss = F.cross_entropy(v_pred, v_y)

            valid_loss += loss

            _, vpreds = torch.max(v_pred, dim=1)
            accuracy = torch.tensor(torch.sum(vpreds == v_y).item() / len(vpreds))

            valid_acc += accuracy

    val_avg_acc= valid_acc/(v_idx+1)


    print("\n Validation set: itr: {:02d} \t loss: {:.2f} \t avg_acc : {:.2f}".format(itr,valid_loss, val_avg_acc))
```

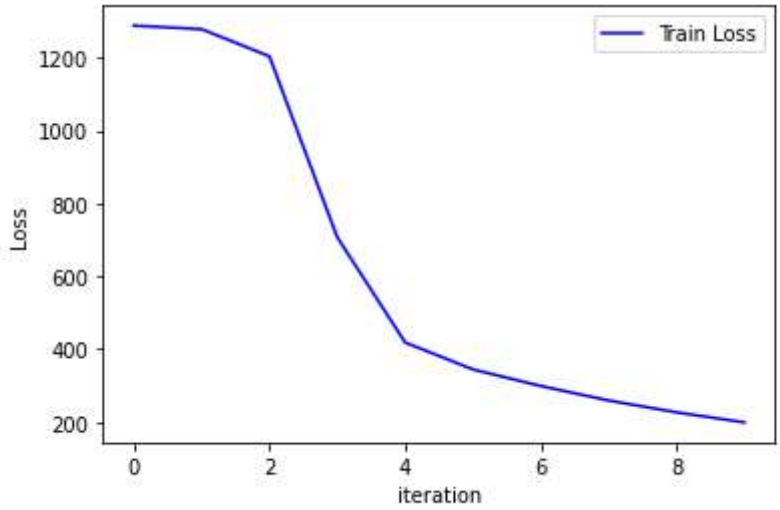
Q7.1.3 (2 points Code+WriteUp)

Results:

```
device = cpu
Get dataset
dataset size train, test
(10800, 1024)
(10800, 36)
itr: 00          loss: 1289.08   acc : 12.57     avg_acc : 0.03
itr: 01          loss: 1279.32   acc : 35.93     avg_acc : 0.10
itr: 02          loss: 1204.66   acc : 100.07    avg_acc : 0.28
itr: 03          loss: 707.36    acc : 183.00    avg_acc : 0.51
itr: 04          loss: 418.51    acc : 240.90    avg_acc : 0.67
itr: 05          loss: 344.35    acc : 261.67    avg_acc : 0.73
itr: 06          loss: 298.77    acc : 274.53    avg_acc : 0.76
itr: 07          loss: 258.97    acc : 285.83    avg_acc : 0.79
itr: 08          loss: 226.53    acc : 294.63    avg_acc : 0.82
itr: 09          loss: 198.86    acc : 301.43    avg_acc : 0.84

 Validation set: itr: 09         loss: 74.62     avg_acc : 0.81
```

Plots:





Code:

```
import numpy as np
import scipy.io
import torch
from torch import nn, optim
import torch.nn.functional as F  # a lower level (compared to torch.nn) interface
from torch.utils.data import Dataset, DataLoader
import torchvision
import torchvision.transforms as transforms
from time import time
import matplotlib.pyplot as plt

# Use GPU if available, otherwise stick with cpu
#use_cuda = torch.cuda.is_available()
torch.manual_seed(123)
#device = torch.device("cuda" if use_cuda else "cpu")
print("device = {}".format(device))

print("Get dataset")
train_data = scipy.io.loadmat('data/nist36_train.mat')
valid_data = scipy.io.loadmat('data/nist36_valid.mat')

train_x, train_y = train_data['train_data'].astype(np.float32), train_data['train_labels']
valid_x, valid_y = valid_data['valid_data'].astype(np.float32), valid_data['valid_labels']

tensor_x = torch.Tensor(train_x).float() # transform to torch tensor
tensor_y = torch.Tensor(train_y).int()

val_x, val_y = torch.Tensor(valid_x).float(), torch.Tensor(valid_y).int()

trainset_loader = DataLoader(TensorDataset(tensor_x, tensor_y), batch_size= batch_size, shuffle=True, num_workers=1)
validset_loader = DataLoader(TensorDataset(val_x, val_y), batch_size= batch_size, shuffle=True, num_workers=1)

print("dataset size train, test")
print(train_x.shape)
print(train_y.shape)

# YOUR CODE HERE
#raise NotImplementedError()

max_iters = 10
train_accnist =[]
train_lossnist =[]

# Defining the CNN
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
```

```
# Defining the CNN
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.lay1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=8,
                kernel_size=3,
                stride=1,
                padding=1,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.lay2 = nn.Sequential(
            nn.Conv2d(8, 16, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output  classes 36
        self.fc1 = nn.Linear(1024, 256)
        self.fc2 = nn.Linear(256, 36)
    def forward(self, x):
        #print(x.size(0),x.size(1),x.size(2),x.size(3))
        x = self.lay1(x)
        #print(x.size(0),x.size(1),x.size(2),x.size(3))
        x = self.lay2(x)
        # flatten the output of conv2 to (batch_size, 32 *16*8)
        #print(x.size(0),x.size(1),x.size(2),x.size(3))
        x = x.view(x.size(0),-1 )
        #print(x.size(0),x.size(1))
        x = self.fc1(x)
        #print(x.size(0),x.size(1))
        x= self.fc2(x)

        return x

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for itr in range(max_iters):
    total_loss = 0

    total_acc =0
    for batch_idx, (data, target) in enumerate(trainset_loader):
        data, target = Variable(data), Variable(target)
        #print(batch_idx)
```

```
    total_acc =0
    for batch_idx, (data, target) in enumerate(trainset_loader):
        data, target = Variable(data), Variable(target)
        #print(batch_idx)
        t_x, t_y =data, target

        #Reshaping to meet the dimension critera of nist
        t_x = t_x.reshape(batch_size,1,32,32)
        #print(t_x.shape)
        #print(t_y.shape)

        #Cross entropy doesnt take hot vectors directly
        t_y = torch.max(t_y, 1)[1]

        #print(t_x.shape)
        #print(t_y)
        prediction = model(t_x)


        #F.nll_loss not working
        loss = F.cross_entropy(prediction, t_y)
        total_loss += loss.item()


        _, preds = torch.max(prediction, dim=1)
        accuracy = torch.tensor(torch.sum(preds == t_y).item() / len(preds))
        total_acc += accuracy

        #backward
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
    avg_acc = total_acc/(batch_idx+1)


    print("itr: {:02d} \t loss: {:.2f} \t acc : {:.2f} \t avg_acc : {:.2f}".format(itr,total_loss,total_acc, avg_acc))
    train_accnist.append(avg_acc)
    train_lossnist.append(total_loss)

model.eval()  # set evaluation mode
valid_loss = 0
valid_acc = 0
with torch.no_grad():
    for v_idx, (data, target) in enumerate(validset_loader):
        v_x, v_y = data, target
```

```
model.eval()  # set evaluation mode
valid_loss = 0
valid_acc = 0
with torch.no_grad():
    for v_idx, (data, target) in enumerate(validset_loader):
        v_x, v_y = data, target
        v_x = v_x.reshape(batch_size,1,32,32)
        v_pred = model(v_x)

        v_y = torch.max(v_y, 1)[1]

        loss = F.cross_entropy(v_pred, v_y)

        valid_loss += loss

        _, vpreds = torch.max(v_pred, dim=1)
        accuracy = torch.tensor(torch.sum(vpreds == v_y).item() / len(vpreds))

        valid_acc += accuracy

val_avg_acc= valid_acc/(v_idx+1)


print("\n Validation set: itr: {:02d} \t loss: {:.2f} \t avg_acc : {:.2f}".format(itr,valid_loss, val_avg_acc))
```

Q7.1.4 (15 points Code+WriteUp)

Results: As we can see from the CNN, the results we obtain are much closer to 90-95%

```
Correct Ans1  : T O D O L I S T 1 M A K E A T O D O L I S T 1 C H E C K O F F T H E F I R S T T H I N G O N T O D O L I S T 3 R E A L I Z E Y O U H A V E A L R E A D Y C O M P L E T E D 2 T H I N G S 4 R
E W A R D Y O U R S E L F W I T H A N A P
Obtained Ans1:  T 0 D 0 L I S T I M A K E A T D D 0 L I S T 2 C H E C K 0 F F T T H E F I R S T T H I N G Q N 0 D 0 L I S T 3 R F A L I Z E Y 0 U T H H I A N V E G A L R E A D X C 0 M P L E T E D 2 S 4 R
E W A R D V 0 U R S E L F W I T H A N A P

Correct Ans2  : 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 1 2 3 4 5 6 7 8 9 0
Obtained Ans2:  A B C D E F G H I J K L M N 0 P Q R S T U V W X Y Z I 2 3 4 S 6 7 8 9 Q

Correct Ans3  : H A I K U S A R E E A S Y B U T S O M E T I M E S T H E Y D O N T M A K E S E N S E R E F R I G E R A T O R
Obtained Ans3:  H A I K U S A R E E A S Y B U T S Q M E F I M E S T H E Y D D W T M A K G S E N S E R E F R I G E R A F 0 R

Correct Ans4  : D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G
Obtained Ans4:  D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G
```

Code For CNN:

```
import numpy as np
import scipy.io
import torch
from torch import nn, optim
import torchvision
import torch.nn.functional as F  # a lower level (compared to torch.nn) interface
from torch.utils.data import Dataset, DataLoader
from torchvision.datasets import EMNIST
import torchvision.transforms as transforms
from time import time
from torch.autograd import Variable
# Use GPU if available, otherwise stick with cpu
use_cuda = torch.cuda.is_available()
torch.manual_seed(123)
device = torch.device("cuda" if use_cuda else "cpu")
print("device = {}".format(device))

print("Get dataset")

EMNIST.url = 'http://www.itl.nist.gov/iaui/vip/cs_links/EMNIST/gzip.zip'
# Reference for transform function
# https://stackoverflow.com/a/54513835
transform=torchvision.transforms.Compose([
    lambda img: torchvision.transforms.functional.rotate(img, -90),
    lambda img: torchvision.transforms.functional.hflip(img),
    torchvision.transforms.ToTensor()
])


train_accemnist =[]
train_lossemnist =[]

emnist_train = EMNIST(root="data", split='balanced', train=True, download=True, transform=transform)
emnist_test = EMNIST(root="data", split='balanced', train=False, download=True, transform=transform)


trainset_loader = DataLoader(emnist_train, batch_size=20, shuffle=True, num_workers=0)


testset_loader = DataLoader(emnist_test, batch_size=20, shuffle=True, num_workers=0)
```

```
testset_loader = DataLoader(emnist_test, batch_size=20, shuffle=True, num_workers=0)

# Ref: https://github.com/gaurav0651/emnist/blob/master/train_emnist.ipynb
label_map = ['0','1','2','3','4','5','6','7','8','9',
       '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',
       'a','b','d','e','f','g','h','n','q','r','t']

print(trainset_loader.dataset.data.shape)
print(testset_loader.dataset.data.shape)


class CNNet(nn.Module):
    def __init__(self):
        super(CNNet, self).__init__()
        self.lay1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=3,
                stride=1,
                padding=1,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.lay2 = nn.Sequential(
            nn.Conv2d(16, 32, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer
        #Input dimensions 28*28 convolved into dimension 32*x*x
        self.fc1 = nn.Linear(32*7*7, 1024)
        self.fc2 = nn.Linear(1024, 47)
        #Output layer is 47


    def forward(self, x):
        x = self.lay1(x)
        x = self.lay2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x= self.fc2(x)
        #x = self.fc3(x)
        return x
```

```
        x = self.fc1(x)
        x= self.fc2(x)
        #x = self.fc3(x)
        return x

model3 = CNNet().to(device)
optimizer = optim.SGD(model3.parameters(), lr=0.0021 , momentum=0.9)
max_iters = 5
for itr in range(max_iters):

    model3.train()
    total_loss = 0
    total_acc =0
    for batch_idx, (data, target) in enumerate(trainset_loader):

        data, target = Variable(data), Variable(target)
        #print(batch_idx)
        t_x, t_y =data, target

        #Forward prop
        prediction = model3(t_x)


        #F.nll_loss not working
        #Loss and accuracy
        loss = F.cross_entropy(prediction, t_y)
        total_loss += loss.item()


        _, preds = torch.max(prediction, dim=1)
        accuracy = torch.tensor(torch.sum(preds == t_y).item() / len(preds))
        total_acc += accuracy

        #backward
        loss.backward()
        #Updating
        optimizer.step()
        optimizer.zero_grad()
    avg_acc = total_acc/(batch_idx+1)


    print("itr: {:02d} \t loss: {:.2f} \t acc : {:.2f} \t avg_acc : {:.2f}".format(itr,total_loss,total_acc, avg_acc))
    train_accemnist.append(avg_acc)
    train_lossemnist.append(total_loss)

model3.eval()  # set evaluation mode
valid_loss = 0
```

```
model3.eval()  # set evaluation mode
valid_loss = 0
valid_acc = 0
with torch.no_grad():
    for v_idx, (data, target) in enumerate(testset_loader):

        v_x, v_y = data, target

        v_pred = model3(v_x)



        loss = F.cross_entropy(v_pred, v_y)

        valid_loss += loss
        # Vloss.append(valid_loss)



        _, vpreds = torch.max(v_pred, dim=1)
        accuracy = torch.tensor(torch.sum(vpreds == v_y).item() / len(vpreds))

        valid_acc += accuracy
        # Vaccuracies.append(

#valid_loss /= len(validset_loader.dataset)
val_avg_acc= valid_acc/(v_idx+1)


print("\n Validation set: itr: {:02d} \t loss: {:.2f} \t avg_acc : {:.2f}".format(itr,valid_loss, val_avg_acc))


# YOUR CODE HERE
#raise NotImplementedError()
```

Code for testing on bounding boxes:

```
#Processing the previously appended bounding box images accorind the dataset in the EMNIST
import skimage
import pickle
def process(img):
    process=[]
    for im in img:
        crop_size = skimage.transform.resize(im,(28,28), anti_aliasing=True)
        crop_size = 1 -crop_size
        crop_size = crop_size.T
        #crop_size = crop_size*255
        process.append(crop_size)
    return process
```

```
import numpy as np
import skimage
import skimage.measure
import skimage.color as sc
import skimage.restoration as sr
import skimage.io
import skimage.filters as sf
import skimage.morphology
from skimage.morphology import closing, square
import skimage.segmentation
img1 = pickle.load(open('q4img1.pickle','rb'))
img2 = pickle.load(open('q4img2.pickle','rb'))
img3 = pickle.load(open('q4img3.pickle','rb'))
img4 = pickle.load(open('q4img4.pickle','rb'))
# print(np.asarray(img1).shape)
# print(np.asarray(img2).shape)
# print(np.asarray(img3).shape)
# print(np.asarray(img4).shape)

process1 = process(img1)
process2 = process(img2)
# print(np.asarray(process2).shape)
process3 = process(img3)
#print(np.asarray(process3).shape)
process4 = process(img4)
#print(np.asarray(process4).shape)
```

```
#Funtion to run the CNN and classify the processed images
def ans(process):
```

```
#Funtion to run the CNN and classify the processed images
def ans(process):

    st = ' '
    npimg =torch.Tensor(np.asarray(process).reshape((-1,1,28,28))).float()
    #Reshaped to required dimensions
    v_pred = model3(npimg)
    idx = torch.max(v_pred, 1)[1]
    idx = idx.int()

    id = (idx.numpy()).reshape((-1,1))

    st = ''
    for i in range(id.shape[0]):
        st += label_map[id[i,0]] + ' '
    return st
```

```
import pickle
import string
import numpy as np

Ans1 = 'T O D O L I S T 1 M A K E A T O D O L I S T 1 C H E C K O F F T H E F I R S T T H I N G O N T O D O L I S T 3 R E A L I Z E Y O U H A V E A L R E A D Y C O M P L E T E D 2 T H I N G S 4 R E W A R D
Ans2 = '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 1 2 3 4 5 6 7 8 9 0'
Ans3 = 'H A I K U S A R E E A S Y B U T S O M E T I M E S T H E Y D O N T M A K E S E N S E R E F R I G E R A T O R'
Ans4 = 'D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G'

st1 = ans(process1)
st2 = ans(process2)
st3 = ans(process3)
st4 = ans(process4)
print('\nCorrect Ans1  :', Ans1)
print('Obtained Ans1: ', st1)
print('\nCorrect Ans2  :', Ans2)
print('Obtained Ans2: ', st2)

print('\nCorrect Ans3  :', Ans3)
print('Obtained Ans3: ', st3)

print('\nCorrect Ans4  :', Ans4)
print('Obtained Ans4: ', st4)
```

```
Correct Ans1  : T O D O L I S T 1 M A K E A T O D O L I S T 1 C H E C K O F F T H E F I R S T T H I N G O N T O D O L I S T 3 R E A L I Z E Y O U H A V E A L R E A D Y C O M P L E T E D 2 T H I N G S 4 R
E W A R D Y O U R S E L F W I T H A N A P
Obtained Ans1:  T 0 D 0 L I S T I M A K E A T D D 0 L I S T 2 C H E C K 0 F F T T H E F I R S T T H I N G Q N 0 D 0 L I S T 3 R F A L I Z E Y 0 U T H H I A N V E G A L R E A D X C 0 M P L E T E D 2 S 4 R
E W A R D V 0 U R S E L F W I T H A N A P

Correct Ans2  : 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 1 2 3 4 5 6 7 8 9 0
Obtained Ans2:  A B C D E F G H I J K L M N 0 P Q R S T U V W X Y Z I 2 3 4 S 6 7 8 9 Q

Correct Ans3  : H A I K U S A R E E A S Y B U T S O M E T I M E S T H E Y D O N T M A K E S E N S E R E F R I G E R A T O R
Obtained Ans3:  H A I K U S A R E E A S Y B U T S Q M E F I M E S T H E Y D D W T M A K G S E N S E R E F R I G E R A F 0 R

Correct Ans4  : D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G
Obtained Ans4:  D E E P L E A R N I N G D E E P E R L E A R N I N G D E E P E S T L E A R N I N G
```

Q7.2.1 (10 points WriteUp)

YOUR ANSWER HERE

Appendix: Neural Network Overview

Deep learning has quickly become one of the most applied machine learning techniques in computer vision. Convolutional neural networks have been applied to many different computer vision problems such as image classification, recognition, and segmentation with great success. In this assignment, you will first implement a fully connected feed forward neural network for hand written character classification. Then in the second part, you will implement a system to locate characters in an image, which you can then classify with your deep network. The end result will be a system that, given an image of hand written text, will output the text contained in the image.

Basic Use

Here we will give a brief overview of the math for a single hidden layer feed forward network. For a more detailed look at the math and derivation, please see the class slides.

A fully-connected network $\textbf{f}$, for classification, applies a series of linear and non-linear functions to an input data vector $\textbf{x}$ of size $N\times 1$ to produce an output vector $\textbf{f}(\textbf{x})$ of size $C\times 1$, where each element i of the output vector represents the probability of $\textbf{x}$ belonging to the class i. Since the data samples are of dimensionality N, this means the input layer has N input units. To compute the value of the output units, we must first compute the values of all the hidden layers. The first hidden layer *pre-activation* $\textbf{a}^{(1)}(\textbf{x})$ is given by

$$\textbf{a}^{(1)}(\textbf{x}) = \textbf{W}^{(1)}\textbf{x} + \textbf{b}^{(1)}$$

Then the *post-activation* values of the first hidden layer $\textbf{h}^{(1)}(\textbf{x})$ are computed by applying a non-linear activation function $\textbf{g}$ to the *pre-activation* values

$$\textbf{h}^{(1)}(\textbf{x}) = \textbf{g}(\textbf{a}^{(1)}(\textbf{x})) = \textbf{g}(\textbf{W}^{(1)}\textbf{x} + \textbf{b}^{(1)})$$

Subsequent hidden layer ($1 < t \leq T$) pre- and post activations are given by:

$$\textbf{a}^{(t)}(\textbf{x}) = \textbf{W}^{(t)}\textbf{h}^{(t-1)} + \textbf{b}^{(t)}$$

$$\textbf{h}^{(t)}(\textbf{x}) = \textbf{g}(\textbf{a}^{(t)}(\textbf{x}))$$

The output layer *pre-activations* $\textbf{a}^{(T)}(\textbf{x})$ are computed in a similar way

$$\textbf{a}^{(T)}(\textbf{x}) = \textbf{W}^{(T)}\textbf{h}^{(T-1)}(\textbf{x}) + \textbf{b}^{(T)}$$

and finally the \emph{post-activation} values of the output layer are computed with $$\textbf{f}(\textbf{x}) = \textbf{o}(\textbf{a}^{(T)}(\textbf{x})) = \textbf{o}(\textbf{W}^{(T)}\textbf{h}^{(T-1)}(\textbf{x}) + \textbf{b}^{(T)})$$

where $\textbf{o}$ is the output activation function. Please note the difference between $\textbf{g}$ and $\textbf{o}$! For this assignment, we will be using the sigmoid activation function for the hidden layer, so: $$\textbf{g}(y) = \frac{1}{1+\exp(-y)}$$ where when $\textbf{g}$ is applied to a vector, it is applied element wise across the vector.

Since we are using this deep network for classification, a common output activation function to use is the softmax function. This will allow us to turn the real value, possibly negative values of $\textbf{a}^{(T)}(\textbf{x})$ into a set of probabilities (vector of positive numbers that sum to 1). Letting $\textbf{x}_i$ denote the ith element of the vector $\textbf{x}$, the softmax function is defined as: $$\textbf{o}_i(\textbf{y}) = \frac{\exp(\textbf{y}_i)}{\sum_j \exp(\textbf{y}_j)}$$

Samples from NIST Special 19 dataset

Gradient descent is an iterative optimisation algorithm, used to find the local optima. To find the local minima, we start at a point on the function and move in the direction of negative gradient (steepest descent) till some stopping criteria is met.

Backprop

The update equation for a general weight $W^{(t)}_{ij}$ and bias $b^{(t)}_i$ is $$ W^{(t)}_{ij} = W^{(t)}_{ij} - \alpha*\frac{\partial L_{\textbf{f}}}{\partial W^{(t)}_{ij}}(\textbf{x})\hspace{1cm} b^{(t)}_{i} = b^{(t)}_{i} - \alpha*\frac{\partial L_{\textbf{f}}}{\partial b^{(t)}_{i}}(\textbf{x}) $$ α is the learning rate. Please refer to the back-propagation slides for more details on how to derive the gradients. Note that here we are using softmax loss (which is different from the least square loss in the slides).

References

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010. http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf (http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf).

[2] P. J. Grother. Nist special database 19 – handprinted forms and characters database. https://www.nist.gov/srd/nist-special-database-19 (https://www.nist.gov/srd/nist-special-database-19), 1995.