# CSE 574: Introduction to Machine Learning
Fall 2017

# Project 3: Classification

Instructor: Dr. Sargur N. Srihari

Prasad Salvi
Person #: 50207353

Veerappan Saravanan
Person #:50247314

Sarah Mullin
Person #: 34508498

## Table of Contents

## 1. Project Synopsis

The objective of this project is to classify the images of handwritten digits according to the numeric value of the image. Images represent 10 numbers ranging from 0 to 9. These images are present in two data sets MNIST and USPS. In this project, classification is done using 3 approaches

1. Logistic Regression
2. Single Layer Neural Network
3. Convolutional Neural Network

## 2. Data Partition

For the tasks in the project, there are two datasets given, MNIST dataset and US Postal Service data.

**MNIST Data:**

This data set consists of the images (represented by 784 i.e. 28*28 values for pixel intensities). Each image consists of numeral ranging from 0 to 9. The label in data set is the actual digit in the image. There are 60000 images and 60000 labels in total. The data set is loaded using the 'pickle' library to divide into 3 partitions training, validation and test datasets.

Size of training data set input matrix = 50000*784
Number of labels in training data set = 50000
Size of validation data set input matrix = 10000*784
Number of labels in validation data set = 10000
Size of test data set input matrix = 10000*784
Number of labels in test data set = 10000

```
# Load the MNIST dataset
print("************ Loading MNIST Data ************")
with gzip.open('mnist.pkl.gz', 'rb') as f:
    try:
        train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
    except:
        train_set, valid_set, test_set = pickle.load(f)
f.close()
```

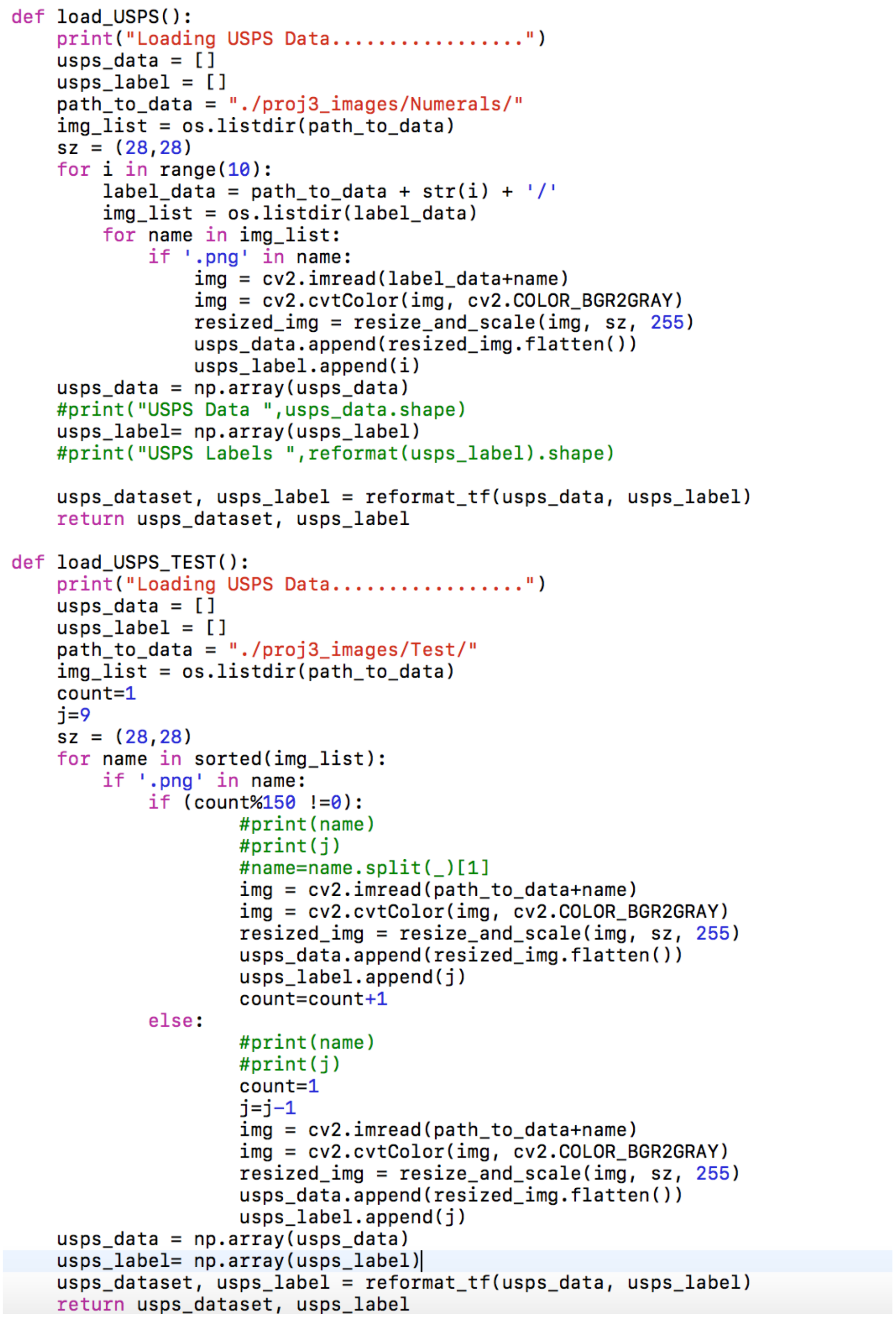
**USPS Data:**

This data set consists of 10 folders of the images (each folder consists of 2000 images, 20000 images in total). Each folder is uniquely named with a digit ranging from 0 to 9. Every folder consists of images of one digit only (i.e. the name of the folder). The label in the data set is the actual digit in the image. The images are loaded from the folder using the os.walk() ,os.path()

functions along with functions from the Python Imaging Library. Each image is reduced into 28*28 image from its actual size using cv2.resize() so that it is represented by 784 feature values. These values are loaded into a numpy array and a label is created using the name of the folder itself. This step is repeated to create

Input data matrix of Dimension = 19999*784 values

Number of labels = 19999



```
def load_USPS():
    print("Loading USPS Data.................")
    usps_data = []
    usps_label = []
    path_to_data = "./proj3_images/Numerals/"
    img_list = os.listdir(path_to_data)
    sz = (28,28)
    for i in range(10):
        label_data = path_to_data + str(i) + '/'
        img_list = os.listdir(label_data)
        for name in img_list:
            if '.png' in name:
                img = cv2.imread(label_data+name)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                resized_img = resize_and_scale(img, sz, 255)
                usps_data.append(resized_img.flatten())
                usps_label.append(i)
    usps_data = np.array(usps_data)
    #print("USPS Data ",usps_data.shape)
    usps_label= np.array(usps_label)
    #print("USPS Labels ",reformat(usps_label).shape)

    usps_dataset, usps_label = reformat_tf(usps_data, usps_label)
    return usps_dataset, usps_label

def load_USPS_TEST():
    print("Loading USPS Data.................")
    usps_data = []
    usps_label = []
    path_to_data = "./proj3_images/Test/"
    img_list = os.listdir(path_to_data)
    count=1
    j=9
    sz = (28,28)
    for name in sorted(img_list):
        if '.png' in name:
            if (count%150 !=0):
                    #print(name)
                    #print(j)
                    #name=name.split(_)[1]
                    img = cv2.imread(path_to_data+name)
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    resized_img = resize_and_scale(img, sz, 255)
                    usps_data.append(resized_img.flatten())
                    usps_label.append(j)
                    count=count+1
            else:
                    #print(name)
                    #print(j)
                    count=1
                    j=j-1
                    img = cv2.imread(path_to_data+name)
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    resized_img = resize_and_scale(img, sz, 255)
                    usps_data.append(resized_img.flatten())
                    usps_label.append(j)
    usps_data = np.array(usps_data)
    usps_label= np.array(usps_label)
    usps_dataset, usps_label = reformat_tf(usps_data, usps_label)
    return usps_dataset, usps_label
```

**One Hot encoding:**

First, the labels associated with each image should be reshaped such that it is a vector with a "one hot" node and all the other nodes are zero. If the position of element in the vector is equal to the label value, then the value at that position is 1, otherwise it is 0. In any case there are 9 zeros and one value is one. For e.g. digit "4" will be represented in vector as [0, 0, 0, 0, 1, 0, 0, 0, 0, 0].

## 3. Logistic Regression

Multiclass logistic regression can be done two ways: using the 1 of k coding scheme (one versus all) where we transfer a multi-class classification problem into K binary logistic classifier problem, such that the sigmoid function is used. In this way, data may be assigned to more than one class.

For the MNIST and USPS data, the problem surrounds classification of numerals, 0 to 9, that are mutually exclusive. Therefore, we can use the softmax transformation of linear functions (1) of feature variables with activations ($a_k = w_k^T x + b_k$) to find posterior probabilities.

$$p\left(C_k|\mathbf{x}\right) = y_k\left(\mathbf{x}\right) = \frac{\exp\left(a_k\right)}{\sum_j \exp\left(a_j\right)} \qquad (1)$$

```python
def activation(X, W, b):
    return (X.dot(W) + b)
```

```python
def softmax(z):
        e_x = np.exp(z - z.max(axis=1, keepdims=True))
        out = e_x / e_x.sum(axis=1, keepdims=True)
        #out=(np.exp(z.T) / np.sum(np.exp(z), axis=1)).T
        return out
```

Using the 1-of-K coding scheme, we create binary vectors with all of the elements zero except element k, which equals one, for the target vectors $t_n$ with feature vector $x_{n,}$ belonging to class $C_k$ (one-hot encoding).

```python
def one_hot(y, n_labels, dtype):
        mat = np.zeros((len(y), n_labels))
        for i, val in enumerate(y):
            mat[i, val] = 1
        return mat.astype(dtype)
```

Using maximum likelihood to determine the parameters of the model directly, our cross-entropy error function becomes

$$E\left(\mathbf{x}\right) = -\sum_{k=1}^{K} t_k \ln y_k$$

```python
def cross_entropy(softmaxprob, y_target):
    return - np.sum(np.log(softmaxprob) * (y_target), axis=1)
```

Taking the gradient of the error function with respect to the parameter vectors $w_j$, we get

$$\nabla_{\mathbf{w}_j} E(\mathbf{x}) = (y_j - t_j)\,\mathbf{x}$$

```python
 # activations, softmax and diff -> n_samples x n_classes:
activation_input = activation(X[idx], w, b)
softm = softmax(activation_input)
diff = softm - y_enc[idx]

# gradient -> n_features x n_classes
grad = np.dot(X[idx].T, diff)
```

Finally, we can use stochastic gradient descent to find the optimum of the error function and the solutions for $w_{j.}$ (see code). We used random normal generation for initial weight values and a vector of 0's for the initial bias values.

```python
w = rgen.normal(loc=0.0, scale=0.01, size=weights_shape)
b = np.zeros(shape=bias_shape)
```

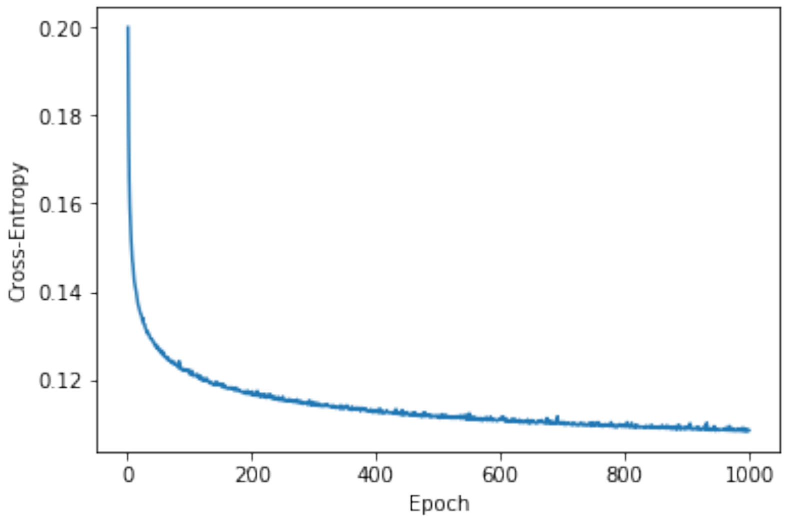
### 3.1. Implementation

Stochastic gradient descent, where for each data point, the cross-entropy error function/cost function was updated, was employed with 1000 epochs and a learning rate, $\eta = 0.001$. Below is a graph of the number of iterations (epochs) on the x-axis and the cross-entropy error function on the y-axis, showing a decreasing trend in the error function as epochs increased, implying minimization of the error function. The MNIST validation dataset was used to find the best from a set learning rate, $\eta = 0.01, 0.001, 0.1$, and the highest accuracy (92.9%) was achieved at $\eta = 0.001$.

```python
print("Accuracy of validation set for eta=0.01", accuracy_01)
print("Accuracy of validation set for eta=0.001", accuracy_001)
print("Accuracy of validation set for eta=0.1", accuracy_1)
```

```
Accuracy of validation set for eta=0.01 (0.91590000000000005, 841, 0.084099999999999994)
Accuracy of validation set for eta=0.001 (0.92930000000000001, 707, 0.070699999999999999)
Accuracy of validation set for eta=0.1 (0.89000000000000001, 1100, 0.11)
```

We note that no form of regularization (such as L1 or L2) to avoid over-fitting was used in this model, since we based the cross-entropy error function off of the formulas provided in the assignment directions.

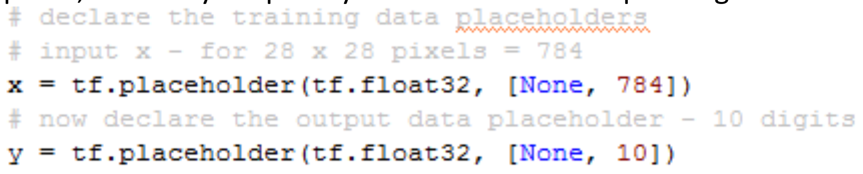


## 4. Single Layer Neural Network (SNN)

Single Layer Neural Network consists of three layers one input layer, one hidden layer and one output layer. Input layer consists of a number of nodes equal to the size of the dataset i.e. rows. Output Layer consists of a number nodes equal to number of classes. In this project it is 10. The number of nodes in the hidden layer and the learning rate should be decided on trial and error basis. Minimum number of nodes in the hidden layer should be a value greater than the number of the features I.e. pixel intensity values, which is equal to 784. Value of learning rate was chosen as 0.001.

### 4.1. Using TensorFlow

TensorFlow requires us to declare the basic structure of the data by using the **tf.placeholder** variable declaration.  Notice the x input layer is 784 nodes corresponding to the 28 x 28 (=784) pixels, and the y output layer is 10 nodes corresponding to the 10 possible digits.



```
# declare the training data placeholders
# input x - for 28 x 28 pixels = 784
x = tf.placeholder(tf.float32, [None, 784])
# now declare the output data placeholder - 10 digits
y = tf.placeholder(tf.float32, [None, 10])
```

There are always L-1 number of weights/bias tensors, where L is the number of layers.  So to setup weights and bias variables, we need to setup two tensors using **tf.Variable()** each. Then, we declare some variables for W1 and b1, the weights and bias for the connections between the input and hidden layer.  This neural network will have 300 nodes in the hidden layer, so the size of the weight tensor W1 is [784, 300].  We initialize the values of the weights using a random normal distribution (**tf.random_normal**) with a mean of zero and a standard deviation of 0.03. Likewise, we create W2 and b2 variables to connect the hidden layer to the output layer of the neural network.

```
# now declare the weights connecting the input to the hidden layer
W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1')
b1 = tf.Variable(tf.random_normal([300]), name='b1')
# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.random_normal([300, 10], stddev=0.03), name='W2')
b2 = tf.Variable(tf.random_normal([10]), name='b2')
```

Then we setup node inputs and activation functions of the hidden layer nodes.

```
# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(x, W1), b1)
hidden_out = tf.nn.relu(hidden_out)
y_ = tf.nn.softmax(tf.add(tf.matmul(hidden_out, W2), b2))
```

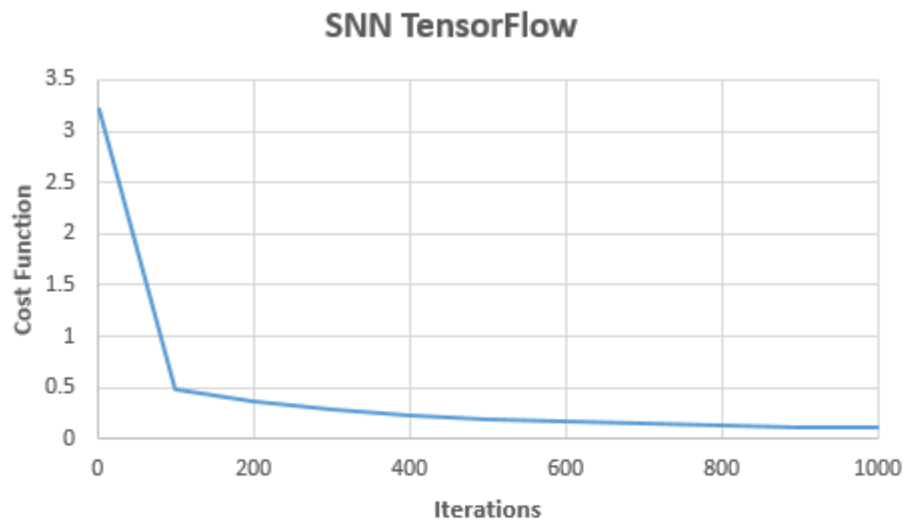
We also have to include a cost or loss function for the optimization / backpropagation to work on. **tf.train.GradientDescentOptimizer()** performs gradient descent and backpropagation.

```
y_clipped = tf.clip_by_value(y_, 1e-10, 0.9999999)
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y * tf.log(y_clipped)
                                              + (1 - y) * tf.log(1 - y_clipped), axis=1))

# add an optimiser
optimiser = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cross_entropy)
```

The correct prediction operation correct prediction makes use of the **tf.equal** function which returns True or False depending on whether to arguments supplied to it are equal.
The **tf.argmax** function is the same as the numpy argmax function, which returns the index of the maximum value in a vector / tensor.
We then want to calculate the mean accuracy from this tensor – first, we have to cast the type of the correct prediction operation from a Boolean to a TensorFlow float in order to perform the **tf.reduce_mean** operation.

```
# define an accuracy assessment operation
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```



### 4.2. Using Backpropagation

Weight matrix W_J represents weight values between the input layer and hidden layer.
Dimension is 784*M, where M = number of nodes in the hidden layer.
Weight matrix W_K represents weight values between the hidden layer and output layer
Dimension is M*N M = number of nodes in the hidden layer, N= number of classes i.e. 10
We initialized these matrices with random weights and normalize.
Bias_1 vector dimension 784*1
Bias_2 vector dimension 1*N (N=10)
In each iteration:

- W_J is multiplied with input data and the resultant value is passed into Logistic sigmoid function to calculate Z_J where Z_J is the activation of hidden layer.

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

- Calculate A_K value (activation value) by multiplying Z_J with W_K.

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j$$

- The predicted value is calculated using

$$y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

- **<u>Cross Entropy Error</u>**
  Then we calculate the cross entropy error between calculated label and the assigned labels.

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k$$

- **<u>Back Propagation</u>**
  - The difference between predicted values and assigned labels gives delta_K

$$\delta_k = y_k - t_k$$

  - The gradient ‘G2’ is obtained by taking product of delta_K and Z_J matrix which is a then multiplied with scalar ‘eta’ (learning rate).

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

- The derivative of h(A_K) is calculated as h(A_K)*(1 – h(A_K)) as logistic regression is considered. W_K and delta_K are multiplied to give a value say temp. The derivative and temp are multiplied to give delta_J vector.

$$\delta_j = h'(z_j) \sum_{k=1}^{K} w_{kj} \delta_k$$

- The gradient 'G1' is obtained by taking product of the input data and delta_J which is then multiplied with scalar 'eta' (learning rate).

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

- The difference between W_J and G1 gives newer W_J value
- The difference between W_K and G2 gives newer W_K value
- The predicted value is used to recalculate the weights by approaching in the reverse direction. Hence the name back propagation algorithm

SNN is implemented on the training dataset to get the weights. The weights of training is given as input to the validation dataset and testing dataset to calculate the accuracy.

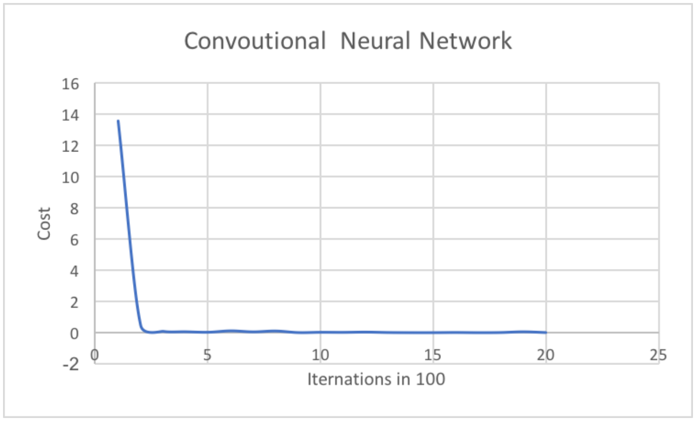
As part of tuning, by trial and error, the above step is repeated for different combinations of Number of nodes in the hidden layer, number of epochs and learning rates and the best combination is chosen. The MNIST validation dataset was used to find the best from a set learning rate, η=0.01,0.001,0.1, and the highest accuracy (97.31%) was achieved at η=0.001.

## 5. Convolutional Neural Network

Convolution Neural Network consists of six layers: one input layer, two convolutional layers, one densely connected layer, one readout layer and one output layer. Input layer consists of a number of nodes equal to the size of the dataset i.e. rows. Output Layer consists of a number nodes equal to number of classes. In this project since we are predicting hand written digits it is 10. The number of nodes in each layer and the learning rate should be decided on trial and error basis. Convolutional layers apply a convolution operation to the input and passes the result to the next layer. In general, a convolution is followed by pooling which keeps only the max value in the neighborhood (down sampling). By doing pooling we reduce the number of features which saves storage space and gives you some sort of local invariance. We could have as many convolution and pooling layers and the output of the final convolution layer is fed into a fully connected layer with user specified neurons and then to a classification layer such as softmax. In-between these two layers we can also choose to have a dropout layer which regularizes the model to prevent overfitting in neural network.

### 5.1. Implementation

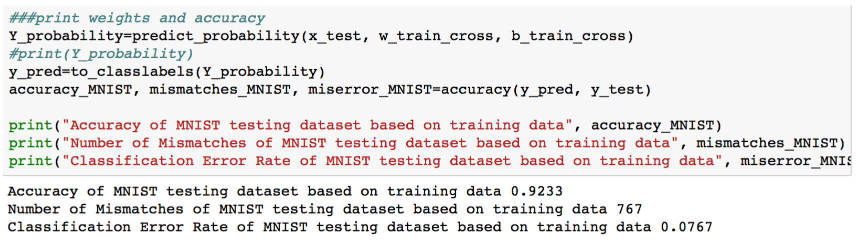
The input image is of 28x28 pixel and hence we have 784 input features. Since we are dealing with a grey scale image we have only one channel per image and it is converted to 4D tensor of size [1,28,28,1]. We use a 5x5 filter to convolute the image using 32 filters to get 32 output channels in the first convolution layer and also reduce the image size to 14x14 by max pooling. The output is fed into the second convolution layer which outputs 64 channels and the pooling layer reduces the image size to 7*7. Now we have 7*7*64 features which is inputted to the fully connected layer having 1024 neurons and the output is passed to the output layer and then softmax is applied for classification. We have used AdamOptimizer for reducing the cross entropy error and used a learning rate of 0.0001.



## 6. Results

### 6.1. Logistic Regression

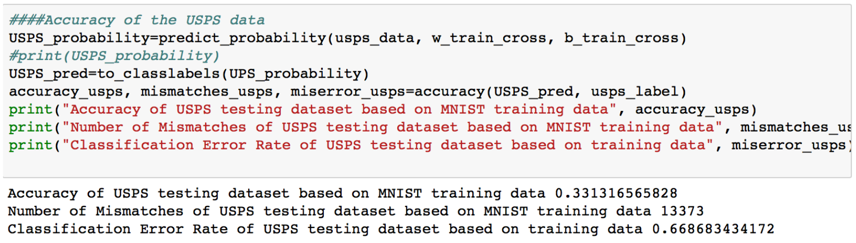
Accuracy MNIST TEST Data: 92.33 %
Classification Error Rate: 0.0767 or 7.67%
Accuracy USPS Numerals Data: 33.13%
Accuracy USPS Testing Data: 32.43%
Classification Error Rate: 0.669 or 66.9%



```
###print weights and accuracy
Y_probability=predict_probability(x_test, w_train_cross, b_train_cross)
#print(Y_probability)
y_pred=to_classlabels(Y_probability)
accuracy_MNIST, mismatches_MNIST, miserror_MNIST=accuracy(y_pred, y_test)

print("Accuracy of MNIST testing dataset based on training data", accuracy_MNIST)
print("Number of Mismatches of MNIST testing dataset based on training data", mismatches_MNIST)
print("Classification Error Rate of MNIST testing dataset based on training data", miserror_MNIS
```

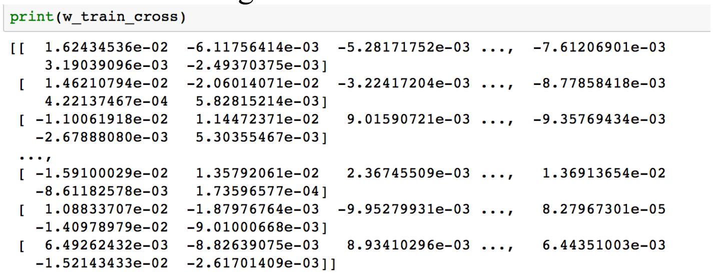
```
Accuracy of MNIST testing dataset based on training data 0.9233
Number of Mismatches of MNIST testing dataset based on training data 767
Classification Error Rate of MNIST testing dataset based on training data 0.0767
```



```
####Accuracy of the USPS data
USPS_probability=predict_probability(usps_data, w_train_cross, b_train_cross)
#print(USPS_probability)
USPS_pred=to_classlabels(UPS_probability)
accuracy_usps, mismatches_usps, miserror_usps=accuracy(USPS_pred, usps_label)
print("Accuracy of USPS testing dataset based on MNIST training data", accuracy_usps)
print("Number of Mismatches of USPS testing dataset based on MNIST training data", mismatches_us
print("Classification Error Rate of USPS testing dataset based on training data", miserror_usps)
```

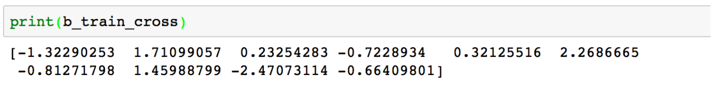
```
Accuracy of USPS testing dataset based on MNIST training data 0.331316565828
Number of Mismatches of USPS testing dataset based on MNIST training data 13373
Classification Error Rate of USPS testing dataset based on training data 0.668683434172
```

The estimated weights and bias are as follows:



```
print(w_train_cross)
```

```
[[  1.62434536e-02  -6.11756414e-03  -5.28171752e-03 ...,  -7.61206901e-03
    3.19039096e-03  -2.49370375e-03]
 [  1.46210794e-02  -2.06014071e-02  -3.22417204e-03 ...,  -8.77858418e-03
    4.22137467e-04   5.82815214e-03]
 [ -1.10061918e-02   1.14472371e-02   9.01590721e-03 ...,  -9.35769434e-03
   -2.67888080e-03   5.30355467e-03]
 ...,
 [ -1.59100029e-02   1.35792061e-02   2.36745509e-03 ...,   1.36913654e-02
   -8.61182578e-03   1.73596577e-04]
 [  1.08833707e-02  -1.87976764e-03  -9.95279931e-03 ...,   8.27967301e-05
   -1.40978979e-02  -9.01000668e-03]
 [  6.49262432e-03  -8.82639075e-03   8.93410296e-03 ...,   6.44351003e-03
   -1.52143433e-02  -2.61701409e-03]]
```



```
print(b_train_cross)
```

```
[-1.32290253  1.71099057  0.23254283 -0.7228934   0.32125516  2.2686665
 -0.81271798  1.45988799 -2.47073114 -0.66409801]
```

Output: Logistic Regression

## 6.2. Single Layer Neural Network (SNN)

**Using TensorFlow:**

Accuracy MNIST TEST Data: 97.31 %

Classification Error Rate: 2.69%

Accuracy USPS Numeral Data: 47.49 %

Accuracy USPS testing data: 45.6%

Classification Error Rate: 52.51%

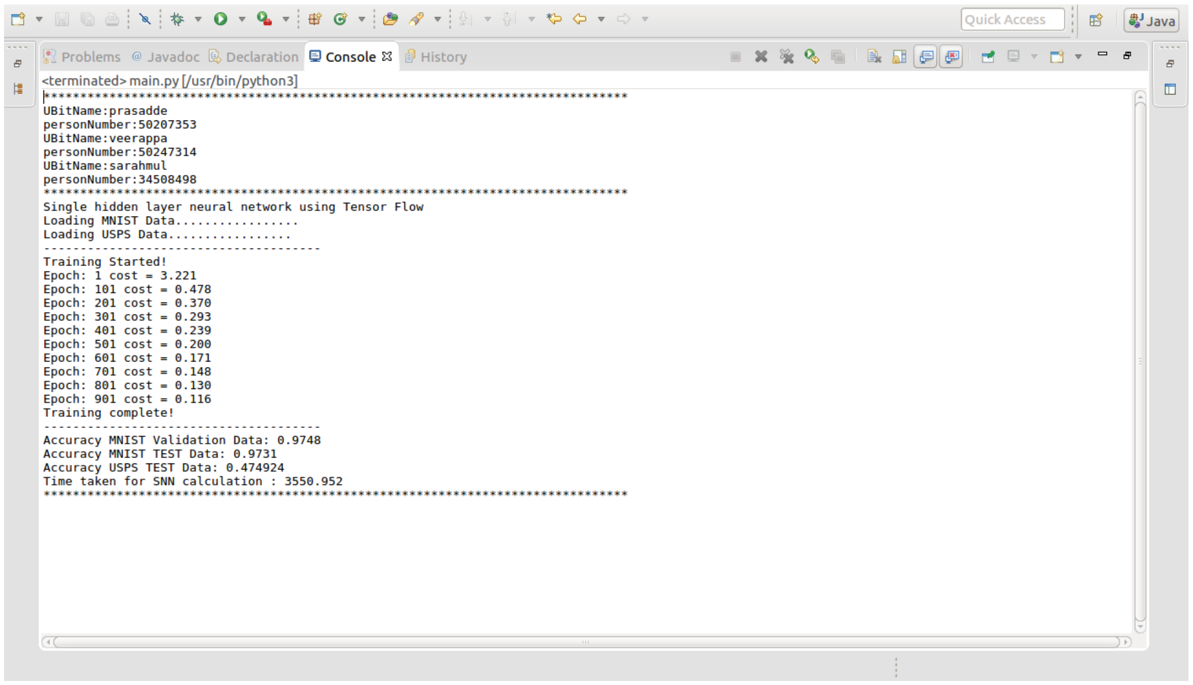
**Using Backpropagation:**

Accuracy MNIST TEST Data: 93.32 %

Classification Error Rate: 6.68%

Accuracy USPS Data: 36.28 %

Classification Error Rate: 63.72%

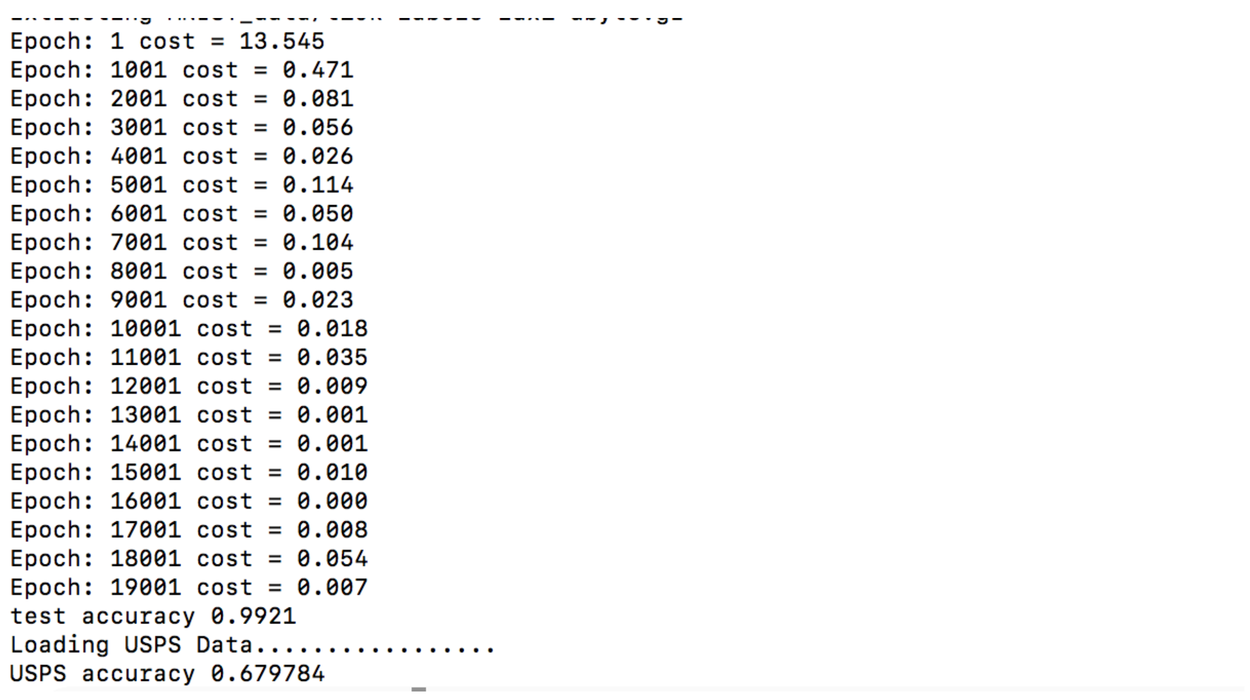


```
<terminated> main.py [/usr/bin/python3]
********************************************************************************
UBitName:prasadde
personNumber:50207353
UBitName:veerappa
personNumber:50247314
UBitName:sarahmul
personNumber:34508498
********************************************************************************
Single hidden layer neural network using Tensor Flow
Loading MNIST Data................
Loading USPS Data................
--------------------------------------
Training Started!
Epoch: 1 cost = 3.221
Epoch: 101 cost = 0.478
Epoch: 201 cost = 0.370
Epoch: 301 cost = 0.293
Epoch: 401 cost = 0.239
Epoch: 501 cost = 0.200
Epoch: 601 cost = 0.171
Epoch: 701 cost = 0.148
Epoch: 801 cost = 0.130
Epoch: 901 cost = 0.116
Training complete!
--------------------------------------
Accuracy MNIST Validation Data: 0.9748
Accuracy MNIST TEST Data: 0.9731
Accuracy USPS TEST Data: 0.474924
Time taken for SNN calculation : 3550.952
********************************************************************************
```

Output SNN TensorFlow

## 6.3. Convolutional Neural Network(CNN)

Accuracy MNIST TEST Data: 99.21 %
Classification Error Rate: 0.79%
Accuracy USPS Numeral Data: 68.22%
Accuracy USPS testing data: 70.33%
Classification Error Rate: 31.78%

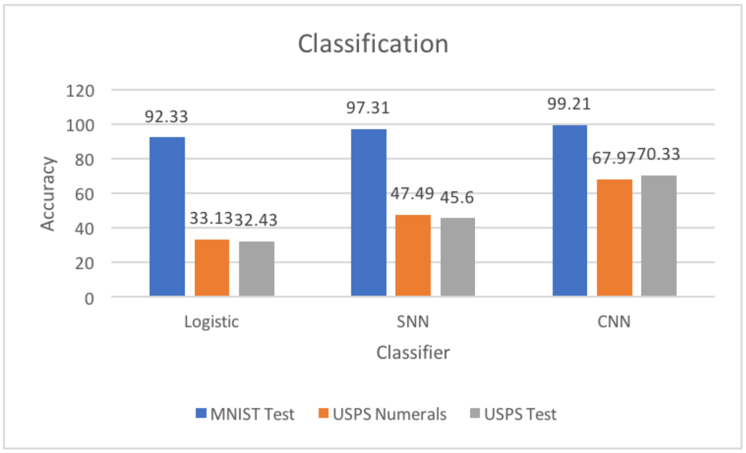


```
Epoch: 1 cost = 13.545
Epoch: 1001 cost = 0.471
Epoch: 2001 cost = 0.081
Epoch: 3001 cost = 0.056
Epoch: 4001 cost = 0.026
Epoch: 5001 cost = 0.114
Epoch: 6001 cost = 0.050
Epoch: 7001 cost = 0.104
Epoch: 8001 cost = 0.005
Epoch: 9001 cost = 0.023
Epoch: 10001 cost = 0.018
Epoch: 11001 cost = 0.035
Epoch: 12001 cost = 0.009
Epoch: 13001 cost = 0.001
Epoch: 14001 cost = 0.001
Epoch: 15001 cost = 0.010
Epoch: 16001 cost = 0.000
Epoch: 17001 cost = 0.008
Epoch: 18001 cost = 0.054
Epoch: 19001 cost = 0.007
test accuracy 0.9921
Loading USPS Data.................
USPS accuracy 0.679784
```

## 7. Conclusions

Classification error rate was the best for the Convolutional Neural Network, especially when generalizing to the USPS dataset. When compared to the tensorflow for the Single Layer Neural Network, there was an increase in 20.73% accuracy when a Convolutional Neural Network was used. Both Single Layer Neural Network and Convolutional Neural Network outperformed, in terms of Classification Error rate and accuracy, a simple multinomial logistic regression model using the softmax function. (See chart)

The model trained using MNIST dataset gave an accurate classifier when tested with its own test data set. But, when tested again a different dataset (like USPS) the accuracy was drastically reduced (less than 50%). This proves the **No Free Lunch Theorem** which states that there is no one model that works best for every problem. The assumptions of a great model for one dataset (MNIST) may not hold for another dataset (USPS).


Classification
Accuracy
Logistic
SNN
CNN
Classifier
■ MNIST Test
■ USPS Numerals
■ USPS Test