Teach a Neural Network to Read Handwriting

- ADWAYT PRADEEP NADKARNI

Problem Statement

Neural networks and deep learning are two success stories in modern artificial intelligence. They've led to major advances in image recognition, automatic text generation, and even in self-driving cars. To get involved with this exciting field, you should start with a manageable dataset.

The **MNIST Handwritten Digit Classification Challenge** is the classic entry point. Image data is generally harder to work with than "flat" relational data. The MNIST data is beginner-friendly and is small enough to fit on one computer. Handwriting recognition will challenge you, but it doesn't need high computational power. Build a neural network from scratch that solves the MNIST challenge with high accuracy.

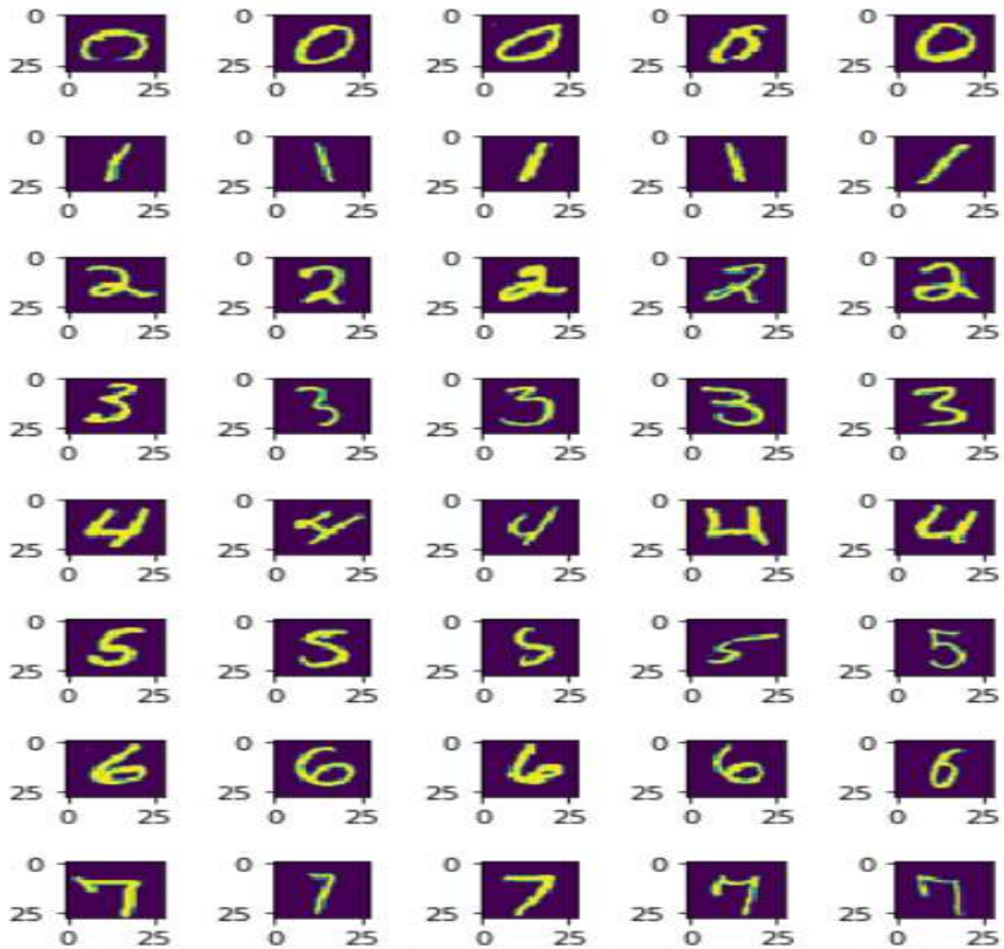
Data Sources **MNIST (http://yann.lecun.com/exdb/mnist/)** – MNIST is a modified subset of two datasets collected by the U.S. National Institute of Standards and Technology. It contains 70,000 labeled images of handwritten digits.

USING ARTIFICIAL NEURAL NETWORK

Algorithm

- Download/Extract the dataset and split the dataset into train, test and validation sets.
- Flatten the images using resize and use one-hot-encoding to give a unique ID to every group of unique digits.
- Set up a neural network architecture to identify the digits
- Check the validation error and model accuracy graph to check the performance of the neural network.
- Test the neural network using arbitrary image.

```python
    for j in range(num_classes):
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, len(x_selected - 1)), :,:])
```



UNIQUE DIGITS

```
]: model = create_model()
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dense_3 (Dense)              (None, 10)                110
_________________________________________________________________
dense_4 (Dense)              (None, 10)                110
=================================================================
Total params: 8,180
Trainable params: 8,180
Non-trainable params: 0
_________________________________________________________________
None
```

NNet Architecture

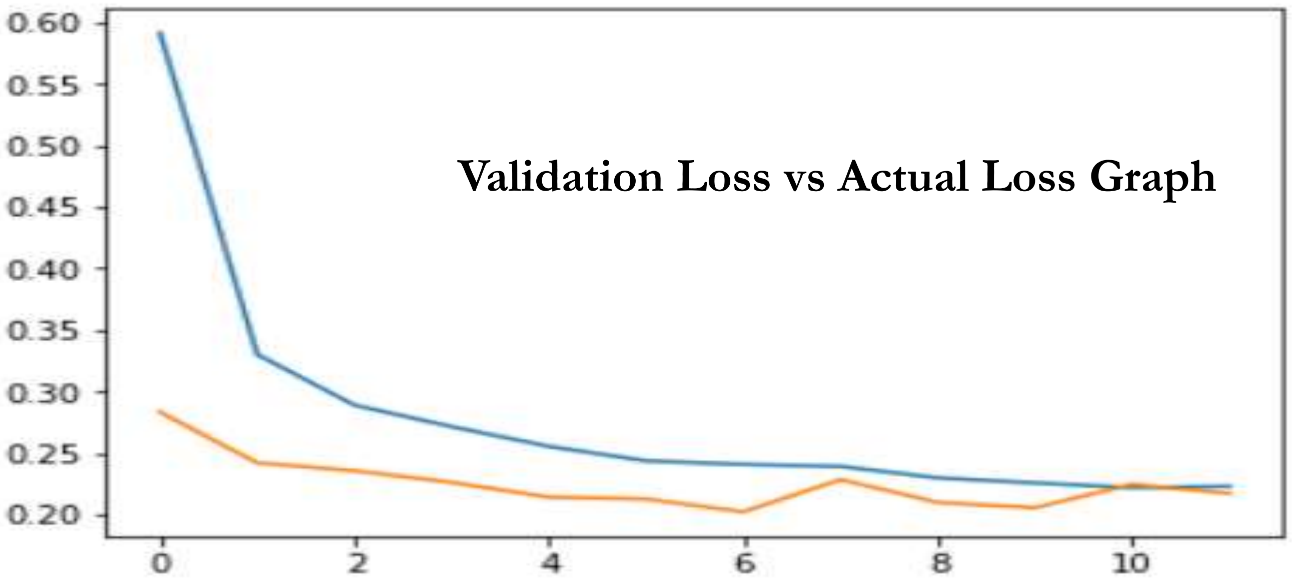
```
In [11]: history= model.fit(X_train,y_train, validation_split=0.1, epochs=12, batch_size=200, verbose = 1, shuffle = 1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/12
54000/54000 [==============================] - 5s 84us/step - loss: 0.5914 - acc: 0.8159 - val_loss: 0.2833 - val_acc: 0.9158
Epoch 2/12
54000/54000 [==============================] - 2s 37us/step - loss: 0.3302 - acc: 0.9039 - val_loss: 0.2420 - val_acc: 0.9317
Epoch 3/12
54000/54000 [==============================] - 2s 36us/step - loss: 0.2889 - acc: 0.9151 - val_loss: 0.2356 - val_acc: 0.9317
Epoch 4/12
54000/54000 [==============================] - 2s 37us/step - loss: 0.2715 - acc: 0.9211 - val_loss: 0.2262 - val_acc: 0.9315
Epoch 5/12
54000/54000 [==============================] - 2s 36us/step - loss: 0.2554 - acc: 0.9254 - val_loss: 0.2141 - val_acc: 0.9387
Epoch 6/12
54000/54000 [==============================] - 2s 36us/step - loss: 0.2437 - acc: 0.9289 - val_loss: 0.2126 - val_acc: 0.9357
Epoch 7/12
54000/54000 [==============================] - 2s 34us/step - loss: 0.2408 - acc: 0.9288 - val_loss: 0.2022 - val_acc: 0.9417
Epoch 8/12
54000/54000 [==============================] - 2s 36us/step - loss: 0.2391 - acc: 0.9292 - val_loss: 0.2285 - val_acc: 0.9325
Epoch 9/12
54000/54000 [==============================] - 2s 39us/step - loss: 0.2299 - acc: 0.9322 - val_loss: 0.2099 - val_acc: 0.9390
Epoch 10/12
54000/54000 [==============================] - 2s 38us/step - loss: 0.2257 - acc: 0.9326 - val_loss: 0.2056 - val_acc: 0.9402
Epoch 11/12
54000/54000 [==============================] - 2s 30us/step - loss: 0.2219 - acc: 0.9339 - val_loss: 0.2247 - val_acc: 0.9325
Epoch 12/12
54000/54000 [==============================] - 2s 31us/step - loss: 0.2230 - acc: 0.9333 - val_loss: 0.2171 - val_acc: 0.9377
```

Final validation accuracy and validation loss after 12 epochs

```
In [12]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
```

```
Out[12]: [<matplotlib.lines.Line2D at 0x2e80e539e48>]
```




```
In [13]: score = model.evaluate(X_test, y_test, verbose=0)
         print('Test score: ' ,score[0])
         print('Test accuracy:', score[1])

         Test score:  0.24280881922841072
         Test accuracy: 0.9279
```

Test score and Accuracy of the NNet

In [14]:

```python
img= img/255
img= img.reshape(1,784)
prediction = model.predict_classes(img)
print('Predicted Digit: ', str(prediction))
```

```
Predicted Digit:  [6]
```

Prediction of the neural network on arbitrary image

USING MACHINE LEARNING

Algorithm

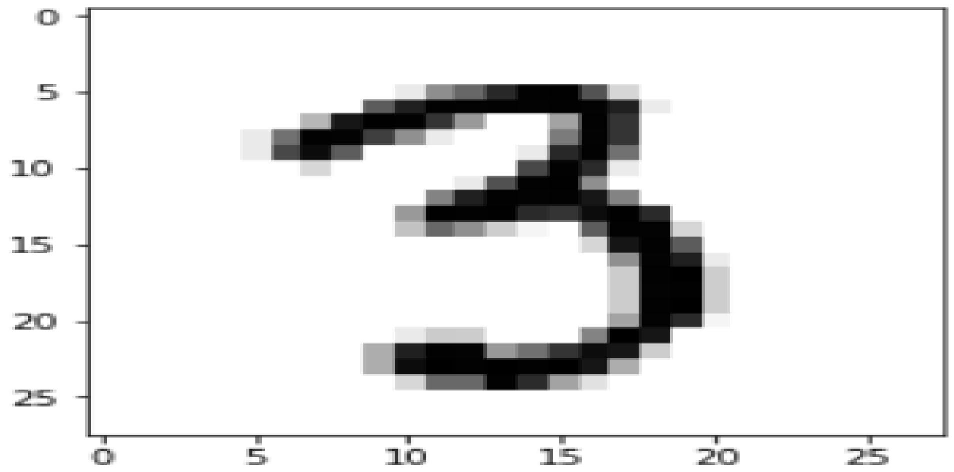
- Download/Extract the dataset(pixels) and split the dataset into features and labels.
- Set up a DecisionTreeClassifier or any classifier of your choice.
- Fit the data into the classifier.
- Check the accuracy by comparing the predicted labels with the true labels
- Test the algorithm using arbitrary image pixels(features).

```
In [3]: clf = DecisionTreeClassifier()
```

```
In [4]: xtrain = data[0:21000,1:]
        train_label = data[:21000,0]
```

```
In [5]: clf.fit(xtrain,train_label)
        xtest = data[21000:,1:]
        actual_label=data[21000:,0]
```

```
In [6]: #check
        d= xtest[5]
        d.shape = (28,28)
        plt.imshow(255-d, cmap='gray')
        plt.show()
```



Number to be predicted

```
In [7]: print(clf.predict([xtest[5]]))

[3]
```

Result

THANK YOU