MNIST Handwritten Digit Recognition

Manish Mohan Kamble

Manishkamble7547@gmail.com

July 2021

1.Application Type

A popular demonstration of the capability of deep learning techniques is object recognition in image data

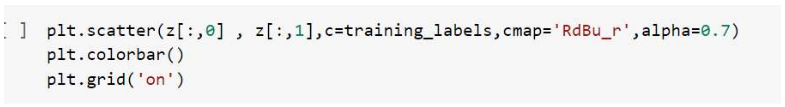
The “hello world” of object recognition for machine learning and deep learning is the MNIST dataset for handwritten digit recognition.

2.About Data Set

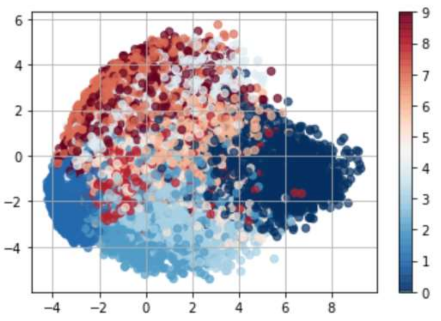
The dataset developed by Yann LeCun , Corinna Cortes and Christopher Burges fro evaluation machine learning models.

The dataset was constructed from a number of scanned document dataset available from the National Institute of Standard and Technology(NIST).Images of digits were taken from a variety of scanned documents, normalized in size and centered. Each image is a 28 by 28 pixel square(784 pixels total).60,000 images are used to train a model and separate set of 10,000 images are used to test it. It a digit recognition task. As such there are 10 digits(0-9) or 10 classes to predict .

We plot scatter plot For better understanding of 10 digits.



```
[ ] plt.scatter(z[:,0] , z[:,1],c=training_labels,cmap='RdBu_r',alpha=0.7)
    plt.colorbar()
    plt.grid('on')
```



3. Loding the MNIST dataset in keras & Preprocessing

The keras deep learning library provides a convenience method for loading the MNIST dataset.

To demonstrate how easy it is to load the MNIST dataset, we will first write a little script to download and visualize the first 4 images in training dataset. You can see that downloading and loading the MNIST dataset is as easy as calling the mnist.load_data() function. We also plot first 9 images using for loop. We print a shape of images which is (60000,28,28) basically it means it is a collection of 60000 images of 28 * 28 pixel.And the pixel intensity range is from 0 to 255.

The training dataset is structured as a 3-dimensional array of instance ,image width and image hight. For 1 hidden layer model we must reduce the image down into vector of pixels. In this case the 28 x 28 sized images will be 784 pixel input values.

We can do this transform easily using the reshape() function on NumPy array.We can also reduce our memory requirements by forcing the precision used by keras anyway.

The pixel values are gray scale between 0 and 255.it Is almost always a good idea to perform some scaling of input values when using neural network models. Because the scale is well known and well behaved, we can very quickly normalize the pixel values to range 0 and 1 by dividing each value by the maximum of 255.

4 . Neural Network

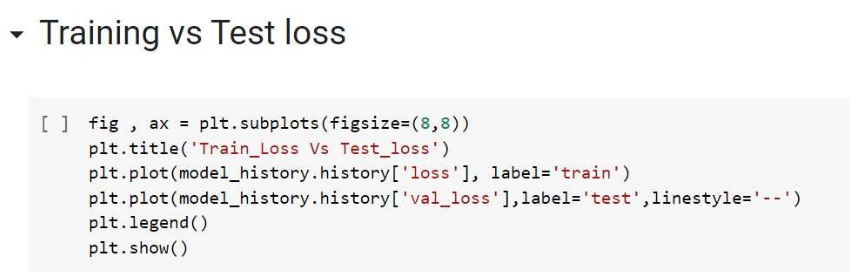
We are now ready to create our simple neural network model. This model is a simple neural network with one hidden layer with the same number of neuron as there are inputs(784).

A "Softmax" activation function is used for the neurons in the hidden layer.

Logarithmic Loss is used as the loss function (called categorical_crossentropy in keras) and efficient ADAM gradient descent algorithm is used to learn the weights.

We can now fit and evaluate the model .The model is fit over 20 epochs and 15 batch size .The test data is used as the validation dataset.

The following chart shows how the training and testing loss decrease with epochs during the training process.

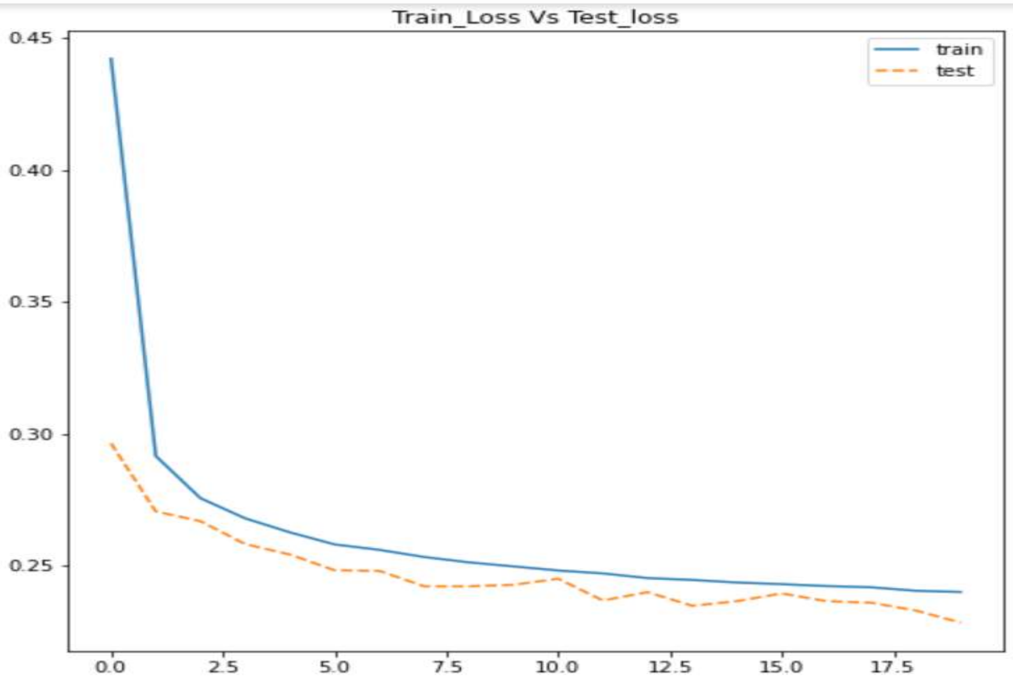


▾ Training vs Test loss

```
[ ] fig , ax = plt.subplots(figsize=(8,8))
    plt.title('Train_Loss Vs Test_loss')
    plt.plot(model_history.history['loss'], label='train')
    plt.plot(model_history.history['val_loss'],label='test',linestyle='--')
    plt.legend()
    plt.show()
```

+ Code   + Text

[ ]

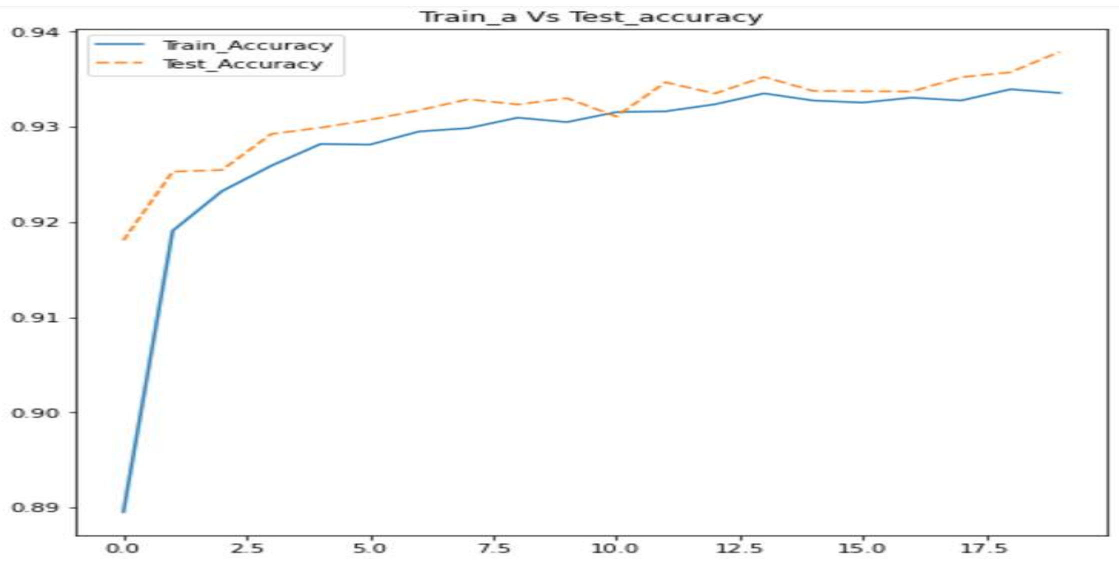


This chart shows how the training and testing accuracy increase with epochs during the training process.

Train_Accuracy vs. Test_Accuracy

```
[ ] fig , ax = plt.subplots(figsize=(8,8))
    plt.title('Train_a Vs Test_accuracy')
    plt.plot(model_history.history['accuracy'], label='Train_Accuracy')
    plt.plot(model_history.history['val_accuracy'],label='Test_Accuracy',linestyle='--')
    plt.legend()
```

[ ]



Finally , the dataset is used to evaluate the model and a classification error rate is printed.

5.Conclusion

You should see the output below .This very simple network defined in very few lines of code achieves a respectable error rate 7.21%. In this notebook we discovered the MNIST handwritting digit recognition problem and deep learning models developed in python using the keras library that are capable of achieving good results.