IMPLEMETATION OF HAND DIGIT RECOGNITION WITH THE MNIST DATASET USING KERAS

Parakram Raj Bharadwaj **Shakamuri Manasa**

AI Technology and Systems

Abstract: Hand digit recognition is one of the important issues that is crucial in order to classify different digits. It is essential to find a technique which recognizes the digits efficiently. Instead of human identifying a digit that is hand written, why cannot a machine do so? Generally computers can identify the digital numbers, but fail to identify hand written numbers. So, to resolve this issue MACHINE LEARNING can be used. By using the current technologies like machine learning, human effort can be reduced to maximum extent. The main focus of this report is to use the algorithms available in Machine learning to design a model which can recognize the hand written digits.

Keywords**:** Artificial Neural Networks (ANN's) Machine learning, Keras, Loss function, Optimizer

1. INTRODUCTION

The definition of hand digit recognition is to make a model or a system identify the digits from any of the sources. Every human being will have their own hand writing. As a human being, we can identify any number or digit in fraction of seconds just by looking at the number once. But to make a machine or a system identify a hand written digit is something unique and important. In bank applications, this new model will be quite helpful and also saves time. Using artificial intelligence and machine learning, we can actually find a better solution to the problem of identifying hand written digits by using a system.

The idea of hand digit recognition can be implemented by using Artificial Neural Networks (ANN). It is one of the computational models which give us the accurate results. A general ANN model consists of the following layers- input layer, hidden layers and output layer. The number of nodes in the input layer is the number of data samples in the dataset. The hidden layers can be one or more depending on the problem statement. Hidden layers are the heart of the ANN model. Any kind of function can be applied on the hidden layers to get the desired output. The final layer is the output layer which consists of the nodes as many as the number of classes to be identified in the given problem. Output layer is simply the result of learning function that is applied. To fetch the results the layers have to be connected and so all the layers in the ANN model are connected to each other.

On the whole, the entire process of identifying the digit is divided into three phases namely creating the model, training the model and testing the model. Each phase of the process yields some result which will be provided as the input to the consecutive layers. The entire processing happens in the hidden layers.

2. EXISTING SYSTEM

The existing system is actually a tedious work because a person has to manually sit and continuously record all the numbers. This may be easy when dealing with few records but when there are millions of records to be updated then this process is not worthy. There is also a chance of human error while detecting the digits because of the light conditions or the paper on which the digits are written may not be so clear. Sometimes human eyesight may not be perfect to identify the numbers.

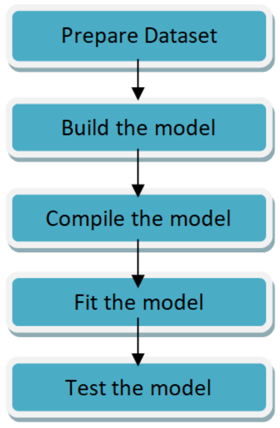
3. PROPOSED SYSTEM

To overcome the issues faced with existing system, we want to develop a system which can identify the digits which are hand written within fraction of seconds accurately. The solution is to make use of

Artificial Neural Networks (ANN's). Artificial Neural Networks can be defined as the set of neurons which simulate the neurons of human brain. A **neural network** is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes [1]. To implement this system, we want to make use of Keras. **Keras** is an open source library written in Python [2]. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives (loss functions), activation functions, optimizers, and a host of tools to make working with image and text data easier.

4. IMPLEMENTATION

To implement any model using Keras, we need a systematic approach. There are 5 phases involved in creating a model. Though we can implement this system using simple machine learning algorithms, we chose neural network model because the accuracy of the neural network model is quite high. As we require large datasets with many samples for a neural network model, the training phase covers wide variety of samples. The following architecture shows the steps to be followed to build a model for hand digit recognition.



4.1 Steps to create a model

4.1 IMPORTING DATASETS:

Firstly, to build a model we require relevant data. The dataset we choose should be appropriate to the problem statement and must contain myriad samples of data. If the numbers of data samples are more, then the model can learn the patterns accurately and can reflect the same during evaluation. For this problem statement we are taking MNIST dataset which contains 70,000 sample images of various digits. To load this dataset, we require suitable packages. The packages are NUMPY, KERAS, and MATPLOTLIB. Each of the packages has some functionality.

NUMPY: This package can be used to perform operations like converting an image into vectors or matrices. (As the dataset contains images as the input samples, we need to convert such samples into matrices)

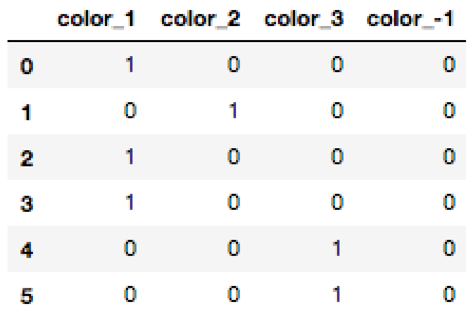
KERAS: It is one of the vital packages because using Keras we can build a neural network model which learns the data samples. Keras is an open source library for building neural network model.

MATPLOTLIB: This package is exclusively used for visualizing the data. The trends and patterns in the dataset can be graphically visualized by using the package.

Next, we need to define few parameters which will be used in creating the neural network model. As we are involved in classifying the digits, we need to know how many digits are there. There are infinite numbers, but all those numbers can contain digits 0-9. So, the number of classes that system/model should learn are 10. The dataset contains 70,000 samples out of which few samples should be taken for training the model and the remaining for evaluating the model. The splitting of samples is choice of the person who is building the model. Usually the training samples are high so as to make the model accurate. Another parameter is **epoch,** which defines the number of iterations required for the model to minimize the **loss function**. A **loss function** or **cost function** is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event [3].

4.2 DATA PREPROCEESING:

It is also called as preparing dataset. The dataset taken need not be necessarily proper which means there may be some uncertainties in the data samples. In order to remove the uncertainties, we must prepare the dataset in such a manner that the model can analyze the samples without ambiguity. The number of classes involved in the problem is 10(0-9 digits). In order to make the model able to classify the digits, represent the digits in the binary form which is known as **one hot encoding**.



	color_1	color_2	color_3	color_-1
0	1	0	0	0
1	0	1	0	0
2	1	0	0	0
3	1	0	0	0
4	0	0	1	0
5	0	0	1	0

4.2.1 Representation of one hot encoding

4.3 BUILD THE MODEL:

The type of the model we are using is **Sequential.** The easiest way to build a model in keras is using Sequential model. A layer by layer model can be built using sequential. Each layer has weights that correspond to the layer the follows it. The model being a neural network, it consists of several layers such as input layer, one or more hidden layers and an output layer. For a neural network model, **Activation function** is quintessential. The main purpose of activation function is to take the input signal and convert it into some output which in turn becomes an input to the next layer. Without an activation function, the output obtained would be a simple linear function. Such linear function may be useful to some extent, but it lacks the ease when dealing with complex function mappings. There are several activation functions available. Among those, we are using **ReLU** activation function for input layers and **Softmax** function for output layer.

4.4 COMPILE MODEL:

Once we are done with the model creation, next step is to compile the model so that we can know about optimization and learning. Compiling the model helps us understand how well the model has learnt the data samples. This is done by using **loss function**. As we are dealing with multi label classification, categorical_crossentropy suits better. Loss function tells us by how much extent the prediction deviates from the actual results (features). This difference can also be called as error. If the deviation is too high, then we must optimize the deviation so that the predictions of the model are accurate. This loss will increase if the prediction deviates from the actual values. The aim of any machine learning model is to improve the loss function so that the difference between the predicted value and the actual value is low. In order to reduce the error in the prediction, we use optimization algorithms

$$\text{CrossEntropyLoss} = -(y_i \log(y'_i) + (1-y_i)\log(1-y'_i))$$

y_i=Actual value (in the dataset)

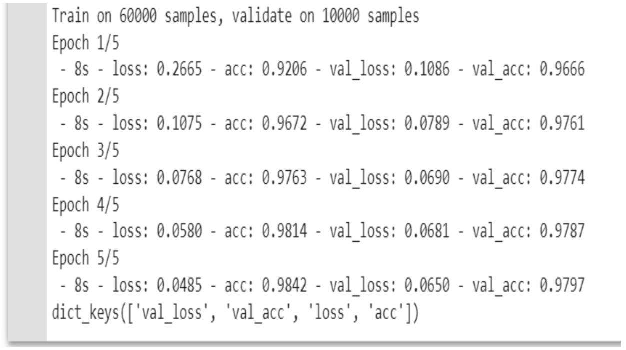
y'_i=Predicted value (from the model)

Now, to optimize the error we require an optimization function (also called as optimizer). There are several optimizers which are available. We can choose one of those algorithms or rather we can check the loss function by using several optimization algorithms and choose the one which optimizes loss function the most. By optimization, we mean the adjustment of the values of weights and bias in such a way that the error minimizes. Here, we are using adam (gradient decent algorithm) as the optimizer to reduce the loss function. Adam is one of the optimization algorithms used to update the weights in an iterative manner until the error is minimized.

4.5 FIT THE MODEL:

Once we create the model, the next phase is to fit the model with the dataset. In order to fit the model, we need the following parameters- training data, test data, batch size, epoch. Training data is the amount of data that we have allocated for training the model. Test data is that part of dataset which is reserved for evaluating the model. As train data and test data are different, we can analyze the performance of the model. The data considered for training contains 60,000 samples and we cannot pass the whole data at a time. So, we fix the batch size which tells us how many data samples to be given for the model each time. Epoch is defined as the number of iterations needed for the network to minimize the loss function, so that it learns the weights..

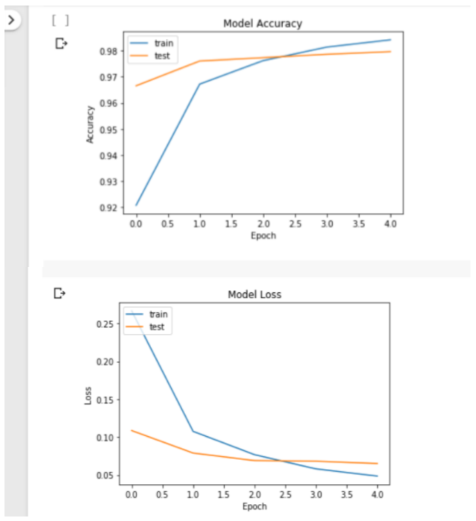
a) USING ADAM OPTIMIZER:



```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
 - 8s - loss: 0.2665 - acc: 0.9206 - val_loss: 0.1086 - val_acc: 0.9666
Epoch 2/5
 - 8s - loss: 0.1075 - acc: 0.9672 - val_loss: 0.0789 - val_acc: 0.9761
Epoch 3/5
 - 8s - loss: 0.0768 - acc: 0.9763 - val_loss: 0.0690 - val_acc: 0.9774
Epoch 4/5
 - 8s - loss: 0.0580 - acc: 0.9814 - val_loss: 0.0681 - val_acc: 0.9787
Epoch 5/5
 - 8s - loss: 0.0485 - acc: 0.9842 - val_loss: 0.0650 - val_acc: 0.9797
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

4.5.1 Loss obtained and accuracy of the model using ADAM optimizer

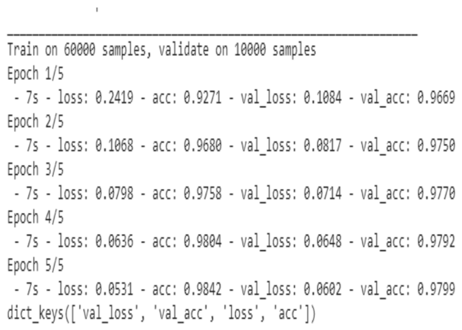
Then, we shall plot the model to analyze the loss and accuracy for a given dataset. Now we can use the package matplotlib to visualize how the given dataset performs during different epochs.



4.5.2 Model Accuracy and Model Loss for ADAM optimizer

We can use any of the available optimizers. Here, we are using 3 of the optimizers. The purpose of using different optimizers is to check which optimizer gives us the maximum accuracy

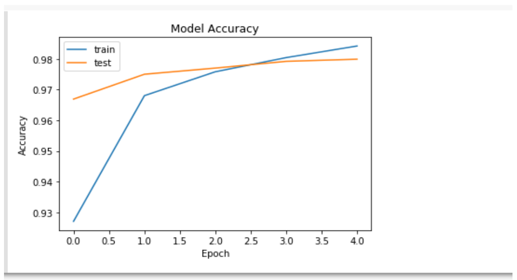
b) **USING ADAGRAD OPTIMIZER**:



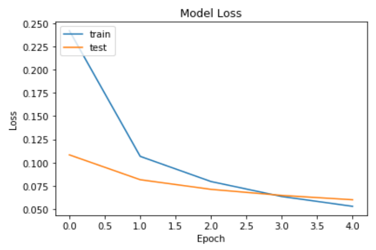
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
 - 7s - loss: 0.2419 - acc: 0.9271 - val_loss: 0.1084 - val_acc: 0.9669
Epoch 2/5
 - 7s - loss: 0.1068 - acc: 0.9680 - val_loss: 0.0817 - val_acc: 0.9750
Epoch 3/5
 - 7s - loss: 0.0798 - acc: 0.9758 - val_loss: 0.0714 - val_acc: 0.9770
Epoch 4/5
 - 7s - loss: 0.0636 - acc: 0.9804 - val_loss: 0.0648 - val_acc: 0.9792
Epoch 5/5
 - 7s - loss: 0.0531 - acc: 0.9842 - val_loss: 0.0602 - val_acc: 0.9799
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

4.5.3 Loss obtained and accuracy of the model using ADAGRAD optimizer

Now, we shall plot the model for accuracy and loss. We observe that the accuracy of the model using ADAGRAD optimizer is same as the accuracy obtained using ADAM optimizer



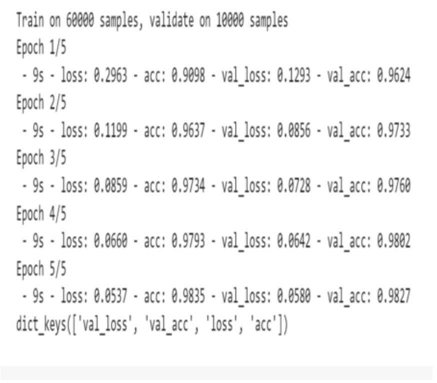
```
# history plot for accuracy
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Model Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(["train", "test"], loc="upper left")
plt.show()
```



4.5.4 Model Accuracy and Loss for ADAGRAD optimizer

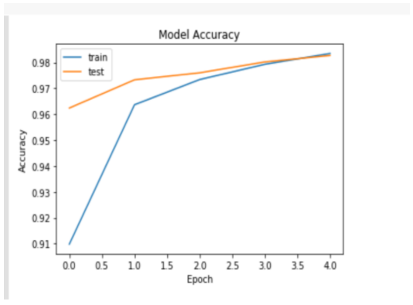
c) USING ADADELTA OPTIMIZER:

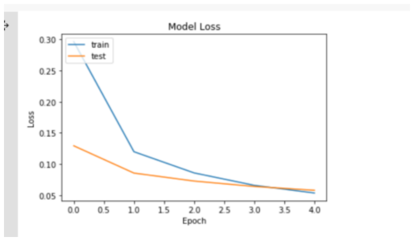
Adadelta optimizer gives us the highest accuracy of 98%. The remaining optimizers might give an accuracy of 99%. But the point is that none of the optimizers give 100% accuracy. So, depending on the requirements for accuracy, we can choose any optimizer.



```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
 - 9s - loss: 0.2963 - acc: 0.9098 - val_loss: 0.1293 - val_acc: 0.9624
Epoch 2/5
 - 9s - loss: 0.1199 - acc: 0.9637 - val_loss: 0.0856 - val_acc: 0.9733
Epoch 3/5
 - 9s - loss: 0.0859 - acc: 0.9734 - val_loss: 0.0728 - val_acc: 0.9760
Epoch 4/5
 - 9s - loss: 0.0660 - acc: 0.9793 - val_loss: 0.0642 - val_acc: 0.9802
Epoch 5/5
 - 9s - loss: 0.0537 - acc: 0.9835 - val_loss: 0.0580 - val_acc: 0.9827
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

4.5.5 Model Accuracy and Loss for ADAGRAD optimizer

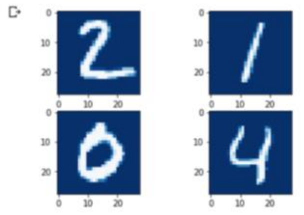




The accuracy and model loss plot of adadelta optimizer shows that adadelta gives an accuracy of 98% which is greater than the prior optimizers chosen. We can test the model with other optimizers as well.

4.6 TEST THE MODEL:

The final step involved in this procedure is testing the model. We have 10,000 samples in the test data. Using the test samples, we can know whether the model built was accurate or not. The figure below shows the output which the model predicted. As there are 10,000 test samples, we have chosen few images to test on the model



4.6.1 Testing the model

5 CONCLUSION:

Hand digit recognition has been one of the issues to be addressed because many of the fields and areas require human written digits to be identified by the system. So, this model can solve the issue of identifying such digits. In spite of many machine learning algorithms; we consider using neural network model because the accuracy is high compared to other algorithms. Using the sequential model, and **adadelta** as the optimizer we could achieve an accuracy of about 98% which is quite high.

6 REFERENCES:

[1]. Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities". Proc. Natl. Acad. Sci. U.S.A. **79** (8): 2554–2558. doi:10.1073/pnas.79.8.2554. PMC 346238

[2]. "Keras backends". keras.io. *Retrieved 2018-02-23.*

[3]. *Wald, A. (1950). Statistical Decision Functions. Wiley.*