Handwritten Character Recognition using CNN

Raghul Shivakumar
Department of Electrical and Computer Engineering
University of Florida
Gainesville, USA
shivakumarraghul@ufl.edu

Rahul Radhakrishnan
Department of Electrical and Computer Engineering
University of Florida
Gainesville, USA
r.radhakrishnan@ufl.edu

***Abstract*—Character Recognition is being used ubiquitously in today's world. Though there are several methods to implement Character Recognition, Convolutional Neural Networks is one of the most common and highly efficient ways to perform it. This paper highlights the implementation of a Convolutional Neural Network and the appropriate selection of the various hyperparameters through careful experimental setup and analyzing the results using cross-validation.**

***Index Terms*—CNN, Pattern Recognition, Gradient descent**

I. Introduction

Handwritten character recognition has proven as a useful tool for processing of documents such as Bank Cheques, tax forms and in applications such as text to speech conversion, retrieving missing contents in a document, etc. There are many well-known techniques with good accuracy for character recognition for printed scripts and hence there is a need to determine an image classification method for handwritten characters. There has been a lot of interest in this area because of the widespread need to digitize handwritten data. The advent of machine learning techniques to classify images has resulted in the adoption of many of those methods for handwritten character recognition. Some machine learning models that have been used are Neural Networks, KNN classifier, RBF (Radial Basis Function) networks, etc and many of them have been proven to produce low error rates [1]. There are 2 common approaches followed in classification to overcome the problems involved with character recognition. One approach involves the use of preprocessing techniques to extract features from images of individual characters and using the features extracted for classification. The disadvantage of this approach is that the model is task specific and hence cannot be generalized. This approach is tedious and requires a lot of human input.The second approach is the use of Convolutional neural networks where the idea is to make the network learn the features required for image classification which is an efficient way to generalize and automate the process eliminating the need for careful feature extraction. Even though this approach has proven to be an efficient method for handwritten character recognition and overcomes the drawback of the first approach, it has drawbacks that are found in the gradient descent algorithm such as getting stuck at a local optima [2]. But, even with this drawback,as shown by Y. Le Cun et al. in [1] large backpropagation networks perform well for image classification and with good experimental design results in a system with high accuracy. In this project, we use the backpropagation algorithm with a gradient descent approach for Convolutional Neural Networks to classify handwritten characters with high accuracy when images of individual characters are given as inputs.CNNs are more robust to scaling and shifting of the character in the image when compared to ordinary Neural Networks. They also have fewer 'weights' when compared to Neural Networks since they are a type of shared weights network. It also makes more intuition to use Convolution because of its prominent use in image processing. The project involves the classification of the characters 'a' and 'b' given in a data set and further the model is generalized to classify between nine different characters namely 'a', 'b', 'c', 'd', 'h', 'i', 'j' and 'k' and 'unknown'.

II. Implementation

We have implemented the Convolutional Neural Network in the Python programming language. We have used python packages and libraries such as numpy, PyTorch, sklearn, etc. We trained two networks for two datasets. The first network classifies images into two classes 'a' and 'b'. The second network classifies images into 9 classes: 'a', 'b', 'c', 'd', 'h', 'i', 'j', 'k', and 'unknown' character.

We have used the scikit package to preprocess the image and the PyTorch package to build the Convolutional Neural Network. In the dataset we used for training there were some images that were rotated. Therefore we changed their orientation and used those images. We also removed invalid images to make our network more efficient. The input images are converted into a 3 dimensional tensor of dimensions $numberofinputimages$ x 50 x 50. We have used the Mean Squared Error (L2 norm) loss function to calculate the loss. We use a threshold to determine whether the output belongs to one of the valid classes or the unknown class.

A. Training and Validation Data

We have split the dataset into training and validation data. The training data consists of three-fourth of the entire dataset and the validation data is made up of the rest. We also shuffled the dataset before separating them into training and validation data.

B. Pseudo-code for training

The below pseudo-code explains the training process.

Algorithm 1 Pseudo-code for the Training

```
Input: raw images, labels
Output: Trained Network
  Initialisation :
  Preprocessing of raw images
     Resize the image shape to $50x50$
     Convert the image from RGB to Gray images
  Define the Neural Network
  Select the hyperparameters, optimization function, Learning
  rate and the loss function
  for $epoch = 1$ to number of epochs do
    for $batch = 1$ to number of batches do
      Forward the input image and predict the class
      Calculate the error using the loss function
    end for
    Backpropogate and adjust the weights to minimize the
    loss function
  end for
  Save the network architecture and its parameters
```

C. Pseudo-code for testing

The below pseudo-code explains the testing process.

Algorithm 2 Pseudo-code for the Training

```
Input: raw images
Output: Predicted classes
  Initialisation :
  Preprocessing of raw images
     Resize the image shape to $50x50$
     Convert the image from RGB to Gray images
  Load the neural network architecture and its weights
  for $image$ in $images$ do
    Forward the input image and predict the class
  end for
```

III. Experimental Design

In the previous section a detailed explanation of the implementation of the model was discussed where the various parameters to be decided or selected were mentioned.In this section, we will discuss the experimental setup for the each of the selected parameters discussed along with the challenges faced during setting the values of parameters and the corresponding results. The experiments are performed on the training and validation set created as mentioned in the previous section and the metrics used for evaluation of the result of each experiment are (i) Accuracy (ii) Confusion matrix. The experimental results have been determined by varying a particular parameter keeping the other parameters as a constant.

A. Size of Input image

We need to reshape the images so that they have the same shape but in doing so care must be taken so that important data has not been lost. To find optimal image resolution, training and validation runs are carried out with input images of resolution of $25x25$, $32x32$, $50x50$. It was noted from the experiment that increase in the resolution increases the accuracy.

TABLE I
Size of Input Image

Size	Accuracy for hard Dataset
$25x25$	70%
$32x32$	79%
$50x50$	85%

From table 1, it can be observed that the accuracy is highest for image size of $50x50$.

B. Number of Feature Maps

The idea behind using Convolution Neural Networks for Handwritten Character Recognition is that we are generating the coefficients of the filter and the number of feature maps represent the number of filters required to extract different distinct features such as horizontal lines, vertical lines,curves, etc in an image of an individual character. With observation of structures of different characters we came to a conclusion that at least 4 filters are required for each character and hence test was carried out for filters sizes around 4.

TABLE II
Feature Maps

# of Feature Maps	Training Accuracy	Test Accuracy
2	84%	81%
4	85%	86%
6	86%	83%

From Table II, it can be observed that 4 feature maps produces the best accuracy.

The figure 1 shows the variation in Training and validation accuracy.

C. Learning Rate

Using an optimal learning rate is an important aspect in training a neural network. If a small learning rate is selected then it takes a large number of iterations for the results to converge with minimal error rate and similarly for a large learning rate the optimal value is missed and hence take a large number of iterations for the result to converge. The learning rate was increased in the order of 10 and the accuracy on the

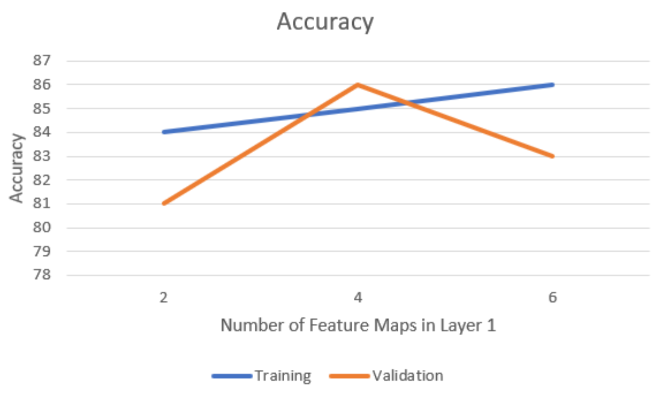


Fig. 1. Accuracy vs # of Feature Maps

TABLE III
LEARNING RATE

Learning Rate	Accuracy for hard Dataset	Accuracy for easy Dataset
0.001	85%	96%
0.01	94%	97%
0.1	95%	98%

validation set was recorded. From Table III it is observed that a learning rate of 0.1 produces the maximum accuracy when all the other parameters are maintained at optimal values.

D. Activation Function

To find the best activation function which produces the best results on the given data set,the network was trained with two activation functions namely tan hyperbolic (tanh) activation function and Rectified Linear Unit (ReLU) activation function. The network training was carried out with different activation functions and the accuracy on the validation set for each of them is listed in the table below.

TABLE IV
ACTIVATION FUNCTION

Size	Accuracy for hard Dataset	Accuracy for easy Dataset
tanh	81%	96%
ReLU	85%	96%

From Table IV, it can be observed that ReLU activation function produced the maximum accuracy among the two activation function.

E. Epochs

Epochs is defined as one pass of the entire data set to be learned by the network through the network layers. When the Epochs is increased the accuracy on the training set increases and hence we need to increase the epochs in a periodic manner to record a fall in the accuracy of the validation data set in order to prevent from overfitting.

Figure 2 represents this variation in the training and test accuracy.

TABLE V
EPOCHS

Epochs	Accuracy for training data	Accuracy for test data
2	68%	67%
4	82%	82%
6	86%	86%
8	88%	84%

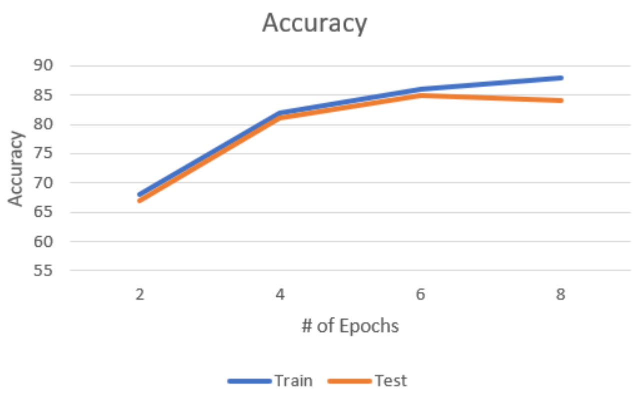


Fig. 2. Plot showing variation between the training and the test accuracy

F. Mini Batch Size

We have used mini batch in our implementation. Using mini batch gives us the combined benefits of both full batch and on-line update methods. We recorded the accuracy for the mini batch sizes 2, 4, and 6 and are presented in the table below.

TABLE VI
MINI BATCH SIZE

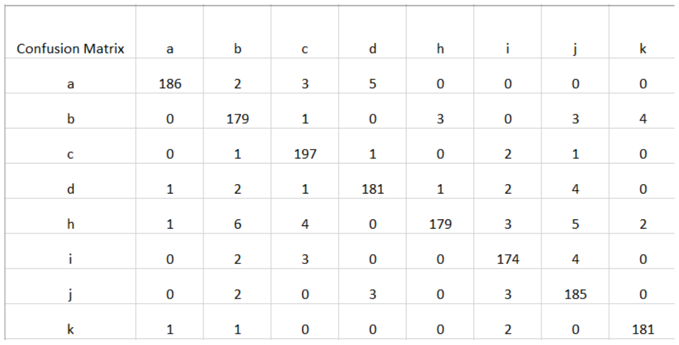
Size	Accuracy for hard Dataset
2	90%
4	85%
6	85%

G. Threshold for identification of class 'unknown'

As explained in the implementation, we set a threshold at the output layer to detect the class unknown. To find an optimal threshold value, a number of sample runs were carried out with a new data set which included handwritten characters other than the eight classes. It was noticed that the output for the valid samples were slightly greater than 0.4 and hence the threshold was set to 0.35 which was found to be good to distinguish outliers from valid characters.

H. Optimal parameters

Based on the above experiments we have selected the following hyperparameters.
Number of feature maps in Layer 1 = 4
Activation Function: Rectified Linear Unit (ReLU)
Number of epochs = 6
Learning Rate = 0.1
Batch Size = 6



Confusion Matrix	a	b	c	d	h	i	j	k
a	186	2	3	5	0	0	0	0
b	0	179	1	0	3	0	3	4
c	0	1	197	1	0	2	1	0
d	1	2	1	181	1	2	4	0
h	1	6	4	0	179	3	5	2
i	0	2	3	0	0	174	4	0
j	0	2	0	3	0	3	185	0
k	1	1	0	0	0	2	0	181

Fig. 3. Confusion Matrix for hard dataset

We obtained an accuracy of 94%.
The confusion matrix in figure 2 represents the performance of our neural network. As we expected the characters 'b' and 'h' are often misinterpreted as each other. Also the characters 'i' and 'j' are misinterpreted as each other. This makes sense because these characters look similar.

IV. Conclusion

From the various experiments conducted, we can conclude that the implementation of convolutional neural networks using back-propagation network and a good experimental design results in a model with a good accuracy and optimised run time and does not involve the complications of extracting features of individual characters through preprocessing techniques. There is still room for improvement with use of deeper networks and complex activation functions with higher accuracy for handwritten character recognition.

References

[1] Lee, Y. (1991). Handwritten Digit Recognition Using K Nearest-Neighbor, Radial-Basis Function, and Backpropagation Neural Networks. Neural Computation, 3(3), pp.440-449.

[2] Seong-Whan Lee, "Off-Line Recognition of Totally Unconstrained Handwritten Numerals Using Multilayer Cluster Neural Network", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No. 6, June 1996.