> <

[1]

Handwritten Character Recognition using a Convolutional Neural Network (December 2019)

T. Gaskin, Undergraduate Student, *University of Florida*,

***Abstract*— The objective of this study was to replicate research done on the effectiveness of convolutional neural networks (CNNs) at the task of classifying handwritten alphabetical characters.**

A CNN was Implemented in order to classify handwritten alphabetical characters in the range a-I (inclusive). Given a dataset of 6400 images: 5120 were used for training the network, and 1280 were used for testing purpose, representing a 80-20 train-test split. After 15 epochs of training, a > 0.990 training accuracy was a achieved and a > 0.910 accuracy was achieved on the test dataset.

I. Introduction

T

HHE effectiveness of CNNs in classification tasks that image processing was discussion in a lecture in the course EEL5840 the Electrical and Computer Engineering Department at the University of Florida [1] . Under further literary investigation, it is a supported idea that CNNs have effectiveness at recognizing patterns in images [2]. Thus it was determined to implement a CNN in order to replicate existing research on the effectiveness of CNNs and to establish an internal reference of the performance of a CNN at the tasks of handwritten character recognition.

II. Implementation

A CNN class was implemented as specified by Phil Tabor in his YouTube tutorial on writing a CNN for handwritten character recognition [3].

The architecture of the network was not changed beyond what was specified by Tabor.

[1]T. Gaskin is with the University of Florida, Gainesville, FL 32601 USA (e-mail: taregaskin@ ufl.edu).

Method showing steps of forward propagation in the network:

```
def forward( self, batch_data ):
        batch_data = torch.tensor( batch_data ).to( self.device
        )
        batch_data = self.conv1(batch_data)
        batch_data = self.bn1(batch_data)
        batch_data = torch.nn.functional.relu(batch_data)
        batch_data = self.conv2(batch_data)
        batch_data = self.bn2(batch_data)
        batch_data = torch.nn.functional.relu(batch_data)
        batch_data = self.conv3(batch_data)
        batch_data = self.bn3(batch_data)
        batch_data = torch.nn.functional.relu(batch_data)
        batch_data = self.maxpool1(batch_data)
        batch_data = self.conv4(batch_data)
        batch_data = self.bn4(batch_data)
        batch_data = torch.nn.functional.relu(batch_data)
        batch_data = self.conv5(batch_data)
        batch_data = self.bn5(batch_data)
        batch_data = torch.nn.functional.relu(batch_data)
        batch_data = self.conv6(batch_data)
        batch_data = self.bn6(batch_data)
        batch_data = torch.nn.functional.relu(batch_data)
        batch_data = self.maxpool2(batch_data)
        batch_data  =  batch_data.view(  batch_data.size()[0],
        -1 )
        classes = self.fc1( batch_data )
        return classes
```

Layer definitions:

```
        self.conv1 = nn.Conv2d(1, 32, 3)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 32, 3)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 32, 3)
        self.bn3 = nn.BatchNorm2d(32)
        self.maxpool1 = nn.MaxPool2d(2)
        self.conv4 = nn.Conv2d(32, 64, 3)
        self.bn4 = nn.BatchNorm2d(64)
        self.conv5 = nn.Conv2d(64, 64, 3)
        self.bn5 = nn.BatchNorm2d(64)
        self.conv6 = nn.Conv2d(64, 64, 3)
        self.bn6 = nn.BatchNorm2d(64)
        self.maxpool2 = nn.MaxPool2d(2)
        self.fc1=nn.Linear(nput_dimensions,
        self.number_of_classes )
```

In addressing prediction on characters not found within the training data: Ideally, characters not present in the training

> <

data can be recognized by setting a threshold on the resulting probability vector. The predicted label of the image given the tensor data will be the highest value above the threshold in the output vector. If there are no values above the threshold in the output vector, then the predicted label will be returned as unknown. This was not implemented.

III. Experiments

If you are using *Word,* use either the Microsoft Equation Editor or the *MathType* add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation *or* MathType Equation). "Float over text" should *not* be selected.

A. Exploring the Dataset To Determine Pre-processing Requirements

Preprocessing was an initial concern, however, an experiment was ran to find the width of the widest image in the dataset and the length of the longest image in the dataset. Given this information, the images before being passed into the CNN were scaled to the dimensions of the length of the longest image in the dataset by the width of the widest image in the dataset.

B. Classification with a Convolutional Neural Network

Given the objective of setting an internal benchmark of the performance of CNNs at classifying handwritten digits, a CNN was Implemented as specified in the implementation section of this document. The network was hyper-parameterized by a learning rate, an epoch limit, and a batch size.

Given a learning rate of .001, a batch_size of 64, an epoch limit of 15, 5120 images were randomly sampled without replacement and fed through the network in batches of 64 in order to train the network. Using a NVIDIA GeForce MX150 with 384 CUDA cores and 4096 MB of GDDR5 memory, training took approximately 141 seconds .

Cross entropy loss was used as the error function. Training performance is summarized as follows:

Figure 1

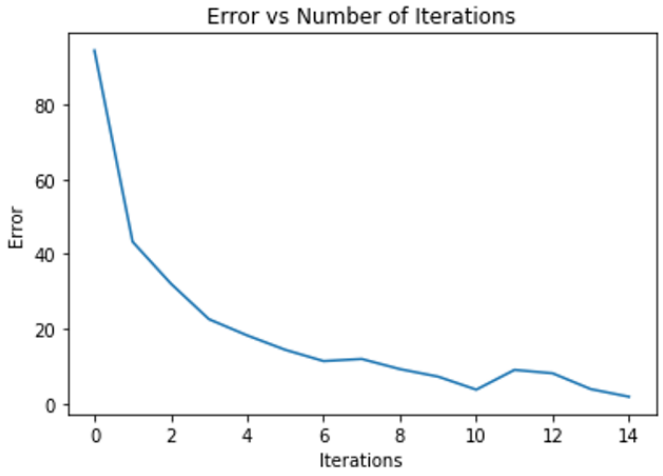
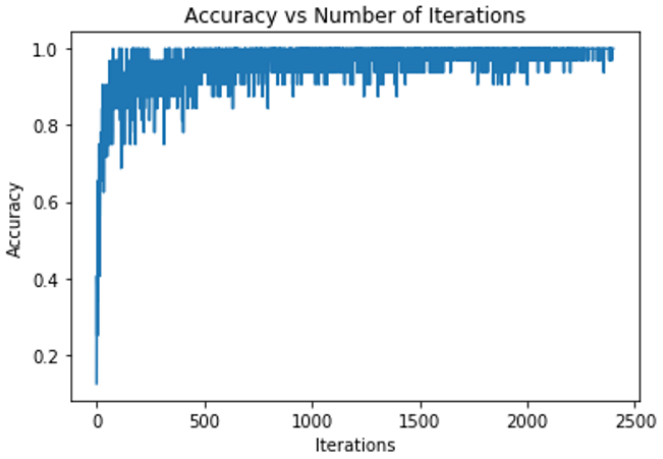


Figure 2



After training, 1280 separate images were used for testing purposes. Test performance summarized below:

Test Accuracy	0.91953125
Test Error	22.469046223908663

IV. Conclusions

A. Internal Benchmark of CNN Performance on Handwritten Character Classification:

The performance of a CNN on handwritten character classification has now been internally benchmarked and can be internally used a reference and/or extended upon in future work.

B. Pytorch as a Simple to Use Framework for Designing Neural Networks

Given the lack of internal experience with PyTorch, it has been observed that PyTorch provides a well-abstracted interface for programming convolutional neural networks.

> <

C. Training Time

The training time of the CNN implement was hypothesized to be between several minutes to several hours. Training took roughly 2 minutes, which was an insightful observance given a low-confidence hypothesized training time.

References

[1] A. Zare. EEL5840. Class Lecture, Topic: "CNN" Department of Electrical and Computer Engineering*, University of Florida, Gainesville, FL Nov., 2019*

[2] Y. Bengio and Y. LeCun, "Convolutional Networks for Images, Speech, and Time-Series," in The handbook of brain theory and neural network, MIT Press Cambridge, MA, 1998. pp 225-258

[3] P. Tabor, "simple-cnn-mnist.py," Sept., 2019. [Online] Available: https://github.com/philtabor/Youtube-Code-Repository/blob/master/simple_cnn_mnist.py [Accessed: Nov 20th 2019]