Handwritten Character Classification Using Convolutional Neural Networks

Gayatri Behera, Ezequiel Juarez Garcia, Joshua Siy, and Junghoon Woo

***Abstract*—The recognition of handwritten characters has been a long-standing challenge for computers. With the use of machine learning, this classification problem has seen improvements. Specific supervised machine learning methods have proven robust to handwriting inconsistency and variability. One such supervised method, convolutional neural networks (CNNs), are at the forefront for handwritten character recognition. In this paper, we use a convolutional neural network to classify a select number of handwritten characters - namely the letters *a, b, c, d, h, i, j*, and *k*. We are able to achieve the classification accuracy of 99.5% for just the letters *a* and *b* and the accuracy of 87% for all of the letters and a class of unknown images.**

***Index Terms*—Handwritten character classification, supervised machine learning, convolutional neural networks.**

I. Introduction

Handwritten character recognition is a research topic that has seen an increased interest in the field of machine learning. While the task of distinguishing characters may be a trivial task for humans, it is not an easy problem for computers. Handwriting by the same person can be inconsistent, meaning that the same letter may not always be written the same way. Shape variability of a character also adds complexity to the problem. Take for example the letter *a*, which can be written in either the single story or double story variant.

Handwritten character recognition enhances the automation of processes and fosters the development of tools that help people reduce repetitive documentation tasks through applications such as mail classification, banking operation, document scanning, and postal address reading [1]. A representative example of this is the conversion of scanned handwritten documents to searchable digital formats. These processes can make the recognition problem more challenging by introducing noise into the character images and reducing the resolution/quality of the characters. It makes finding good features to distinguish characters apart a more complicated task for computers.

Machine learning algorithms can accommodate certain degree imperfections found in real-life classification problems and in recent years have been successfully applied to handwritten character classification. For example, in [2], a deep learning technique is used to classify handwritten digits with a classification accuracy of 98.23%. Due to the complexities mentioned above of handwritten alphabet characters, the classification accuracy of letters is generally not as high as the accuracy for digits.

Researchers have looked into other methods to increase the accuracy of character classification. Convolution neural networks (CNNs) can be a suitable way to increase the accuracy of handwritten characters classification since it is mainly applied to the realm of pattern recognition of the images [3].

We considered many different approaches to aid in the classification process such as support vector machines [4], random forest classifier [5][6], and *k*-nearest neighbors [7][8]. Although the majority of these methods have proven to be a viable solution for unequal or irregular character data, we wanted a chance to test a few hypotheses that could improve the performance of convolutional neural networks (CNNs). Due to the popularity of CNNs [9][10] and the support from the Python community, we were able to focus our efforts on validating or invalidating our hypotheses instead of attempting to implement a CNN from scratch.

The issue with the presented data set was that it involved handwritten samples from different individuals. Parts of them were sourced from the United States Postal Service data and contained many inconsistencies such as overlapping characters, letters written in an unruly manner that even deterred regular, seamless identification from the naked eye, due to the ambiguity involved in correctly assessing the images. The remaining portion of the data set consisted of handwritten cursive letters sourced from students denoting the characters *a, b, c, d, h, i, j*, and *k*. Although the latter category proved to be of better quality in terms of legibility, it also constituted a more significant number of samples, which could lead to overfitting of certain characters during training.

To mitigate some of these issues, we propose two small experiments and one main experiment to improve the performance of the CNNs. The small experiments involved varying the batch size and the learning rate to determine an optimal combination. The main experiment to observe the change in accuracy when evenly sampling images from all of the classes versus not doing that.

The rest of the paper is organized as follows: Section II details out the implementation steps of the classifier. Section III explains the experiments conducted and offers reasoning for the results. Section IV deals with the conclusion and potential improvements that can be made to the algorithm in future revisions.

II. Implementation

The implementation of the CNN for handwritten character classification is separated into two phases: a training phase and a test phase.

A. Training Phase

The most important of the two phases is the training phase. It is where the majority of the design decisions and

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Fig. 1. Equation for neural network output from PyTorch documentation.

experiments take place. In this phase, the training images are imported, processed, and the architecture of the CNN is carefully designed. Afterward, the CNN is trained on a subset of the training data. It is important to use suitable learning parameters and CNN architecture during the training phase to achieve a high classification accuracy.

1) Importing and Processing Images: The combined set of images used for training consists of approximately 6200 letters from our classmates, 1700 images from the original USPS training data set provided by our professor, and 300 *unknown* images sourced from the Internet. The class data and the original data sets are made up of the letters *a, b, c, d, h, i, j*, and *k*. The critical difference between the two sets is the quality of the letters. The class data contains neatly handwritten letters tailored for this project. In contrast, the original data set contains more irregular and ambiguous characters. The *unknown* data set contains images of other letters, digits, landscapes, animals, etc. A difference among all three of these sets is the number of images within them. This difference was taken into consideration in the experiments section.

The images in all of the data sets varied in size. The class data and original data had approximate dimensions of 50×50 pixels. However, the images in the *unknown* data set contained images with larger dimensions. To standardize the dimensions and to comply with the requirements of the PyTorch Python library, all of the images were resized to a dimension of 28×28 pixels. In addition to resizing, the *unknown* images were also converted to binary images to match the images in the class and original data sets. All of these steps are handled by a Python function that outputs a tensor with dimension $N \times C \times H \times W$, where N is the number of images in the batch, C is the number of channels in an image (1 for binary images), H is the height of an image, and W is the width of an image.

2) Constructing CNN: The design of the CNN architecture is one of the most important steps in the training phase. To simplify the process, we used the PyTorch Python library for constructing the CNN. The CNN had to be able to achieve reasonably high accuracy for two types of test sets. One was for classifying solely the letters *a* and *b* and the other one was for classifying the letters *a, b, c, d, h, i, j, k*, and *unknown* images. The former is referred to as the "easy" data set and the latter as the "hard" data set. The formulas in the PyTorch documentation proved very useful for calculating the architecture parameters. One such formula is shown in Figure 1.

3) Training CNN: For testing of our even sampling hypothesis, we used a learning rate α of 0.01, momentum m of 0.5, and a batch size of 32 to train the CNN. These values fall under acceptable ranges as reported in Section III. The CNN architecture previously discussed was used. For the easy and hard data set, the CNN was trained three times and for 300 epochs each time. The reason for training three times was to obtain different randomly initialized variables and different accuracy results. Although all of the combined data was used to train the CNN, only 80% was used in the actual training, and the remaining 20% was used to validate the model. The models are saved as binary files which are loaded back up by the PyTorch library in another Python test script. The results of the training are summarized in Table I.

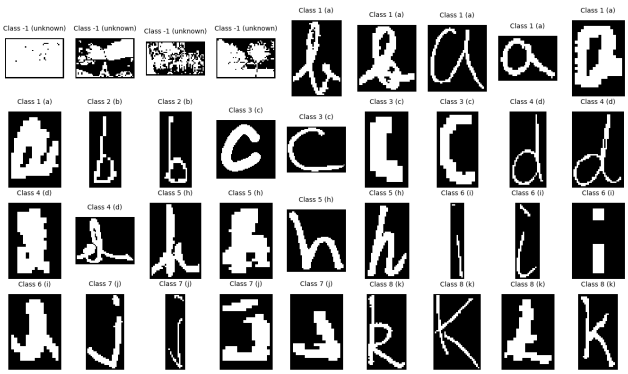


Fig. 2. Character examples drawn from all of the data sets. Some of the images exhibit irregularities and ambiguity that makes the training of a CNN more difficult.

B. Test Phase

The remaining 20% of the training data was used as a test set for the CNN models. Part of the reason why we chose 20% was that we wanted most of the data to be used for training since the test data would only be used for testing. A more even split of the data would have exposed the CNN to far fewer training samples, which can adversely affect its robustness.

Algorithm 1 CNN Training Algorithm

Input: names of training images and labels
Output: model files
 # Initialization
specify CNN architecture and parameters learning rate α, momentum m, number of epochs n, and batch size s
load images and labels
binarize images and resize to 28×28
do a forward pass with randomly initialized values
 # Epochs loop
for $i = 1$ to n **do**
    perform back propagation and dropout
    do a forward pass
end for
validate model

III. Experiments

The main experiment we set out to validate or invalidate was the even sampling of images from each class. However, before performing this experiment, we first wanted to address some other smaller experiments in an effort to squeeze as much of a performance from the CNN as possible.

Before proceeding, we would like to note that we implemented the dropout technique [11] in the second convolutional layer of the CNN as an added protection against overfitting. For the small experiments and generation of training and validation sets experiment, unless otherwise stated, the architecture of the CNN described in Section II is used. The discussion of the experiments is mostly based on the easy data set though certain similarities or differences with the hard data set are also discussed.

A. Small Experiments

The first small experiment was the selection of a suitable learning rate. Generally, the smaller the learning rate, the more computational resources that are required to train a CNN but the closer we can approximate an error local minimum. This diminishing return behavior prompted us to experiment with learning rates in the ranges of 0.01 to 0.3 for the easy data set. In all of the runs, the number of epochs is kept to 12 to reduce the computational time needed to generate the plot. The results are shown in Figures 3(a). As seen in the plot, a learning rate in the range of 0.01 to 0.18 yields the highest accuracy. An α greater than 0.18 fluctuates between accuracy of about 90% and 50%. This behavior signifies that values greater than 0.18 are not suitable for the CNN. The stochastic nature of the CNN training explains the fluctuations.

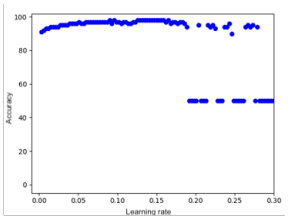
The second experiment we performed was to find a suitable batch size. For this experiment, we fixed the learning rate α to 0.095 which falls in the satisfactory learning rate range we previously observed. The accuracy was measured as the batch size s was varied from 1 to 1412, which was the total number of training images. Batch learning has the benefit of smoother convergence to an error local minimum whereas online learning is prone to more noisy accuracy results but has a higher probability of jumping out of a local minimum. The results are displayed in Figures 3(b) & 3(e). The results of the hard data set exhibited a similar trend as shown in Figures Figures 3(d) & 3(c).

B. Generation of Training and Validation Sets

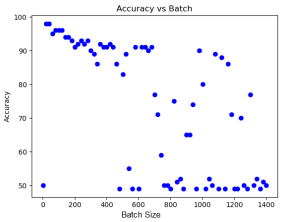
We hypothesized that sampling the same number of images from each class to generate a training set for the CNN would result in higher classification accuracy when compared to using all of the images from each class for the training set. The reasoning behind this hypothesis stems from the fact that we had significantly more images from the class data set than we did from than the original data set, or about 3.5 times as much, and we only had 300 images from the *unknown* data set. These unbalanced data sets could lead the CNN to learn one class more than another.

As discussed in the implementation section 80% of the combined data set, which consisted of about 8,200 images, were used to train the CNN. For the "original" method for training the CNN, we did not take into consideration that amount of images belonging to each class. For the even sampling method, we drew 200 images from class because that was the lowest number of images per class. This meant that we had a total 1,800 images to train with, a drastically smaller number than in the original method. The results shown in Table I show the results of the experiment on the easy data set. Although not included in the table, we were able to obtain an average accuracy of 87% for the hard data set.

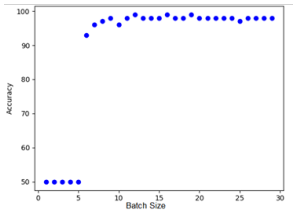
The Final Accuracy column shows that accuracy at the last epoch of the training. Test set 1 and test set 2 consist of approximately 200 randomly sampled images of *a* and 200 of letter *b*. On both test sets, the original method for sampling yield a higher classification CNN than the even sampling method. This invalidates our hypothesis that even sampling is a better method for training the CNN. One possible explanation for this behavior is that the max pooling and dropout used in the design of the CNN is sufficient to prevent overfitting of letters. By using even sampling, we also reduced the amount of training data what the CNN was exposed to. This makes the CNN less robust to samples that it has never encountered.



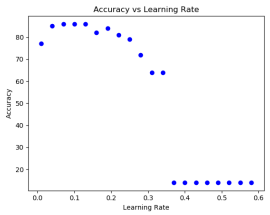
(a) Accuracy as learning rate is varied from 0.01 to 0.30 for easy data set.



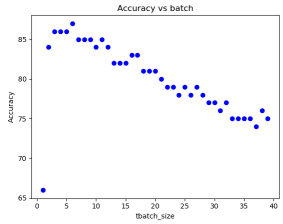
(b) Batch sizes in the range of 1 (online) through 1412 (batch) for easy data set.



(c) Zoomed in plot of (b) with batch sizes in the range of 1 (online) through 30 (mini-batch).



(d) Hard data set's accuracy versus learning rate.

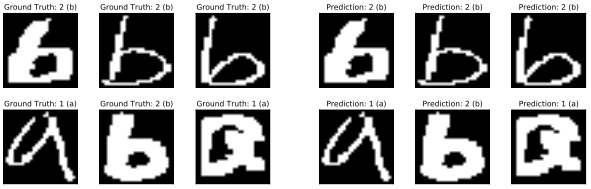


(e) Hard data set's end accuracy versus batch size.

Fig. 3. Experiments varying the learning rate and batch size to determine the value that yields the best classification accuracy.

TABLE I
RESULTS FROM TRAINING AND VALIDATION SET GENERATION EXPERIMENT

Method Type	Final Accuracy (%)	Test Set 1 Accuracy (%)	Test Set 2 Accuracy (%)
Original	99	99.5	97.73
Evenly Sampled	99	85.75	86.12



(a) Ground truth of selected characters from the easy data set.

(b) Predictions of selected characters from the easy data set.

Fig. 4. Ground truth and predictions examples from easy data set.

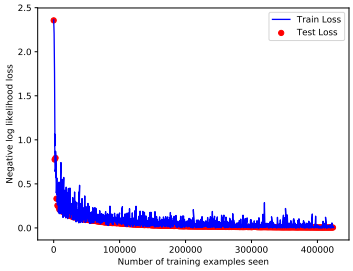


Fig. 5. Classification loss for one of the CNN models for the easy data set. A learning rate of 0.01 and momentum of 0.5 was used in conjunction with a batch size of 32. All of these values are supported by the experiments performed in this section.

IV. CONCLUSION

After assessing the resultant accuracy of the system for both of the data sets, we conclude that training using a mini-batch approach yielded better overall accuracy as this ensured that the model was least affected by bias that would result from varying samples of data sourced from the different data sets. It helped suppress the effects of over-fitting and built up a more robust model that was resilient to any outliers that existed in the underlying sample set.

Also, after experimenting with various classifiers, we narrowed down on CNN as the preferred methodology as it provided a better accuracy owing to back-propagation happening at the end of each iteration that helped minimize the error margin.

ACKNOWLEDGMENT

We would like to thank our classmate Bo Hu for his insights on PyTorch's CNN library, the teaching assistants and professor for answering our questions.

REFERENCES

[1] J. Pradeep, E. Srinivasan, and S. Himavathi, "Neural network based handwritten character recognition system without feature extraction," in *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*, March 2011, pp. 40–44.

[2] H.-M. Jeon, V. D. Nguyen, and J. W. Jeon, "Real-time multi-digit recognition system using deep learning on an embedded system," in *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication*, New York, NY, USA, 2018, pp. 74:1–74:6. [Online]. Available: http://doi.acm.org/10.1145/3164541.3164641

[3] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015. [Online]. Available: http://arxiv.org/abs/1511.08458

[4] P. Sok and N. Taing, "Support vector machine (svm) based classifier for khmer printed character-set recognition," in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, Dec 2014, pp. 1–9.

[5] L. P. Cordella, C. D. Stefano, F. Fontanella, and A. S. D. Freca, "Random forest for reliable pre-classification of handwritten characters," in *2014 22nd International Conference on Pattern Recognition*, Aug 2014, pp. 1319–1324.

[6] S. Bernard, S. Adam, and L. Heutte, "Using random forests for handwritten digit recognition," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Sept 2007, pp. 1043–1047.

[7] A. R. F. Quiros, R. A. Bedruz, A. C. Uy, A. Abad, A. Bandala, E. P. Dadios, A. Fernando, and D. L. Salle, "A knn-based approach for the machine vision of character recognition of license plate numbers," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, Nov 2017, pp. 1081–1086.

[8] T. K. Hazra, D. P. Singh, and N. Daga, "Optical character recognition using knn on custom image dataset," in *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)*, Aug 2017, pp. 110–114.

[9] D. S. Maitra, U. Bhattacharya, and S. K. Parui, "Cnn based common approach to handwritten character recognition of multiple scripts," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, Aug 2015, pp. 1021–1025.

[10] I. Ramadhan, B. Purnama, and S. A. Faraby, "Convolutional neural networks applied to handwritten mathematical symbols classification," in *2016 4th International Conference on Information and Communication Technology (ICoICT)*, May 2016, pp. 1–4.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313