Machine Learning for Hand-Written Image Recognition

Dhruv Kushwaha[*], Shivam Bang[+], Yao An Lee[—], Tre' Jeter[+]
[*]Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32603 USA
[+]Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA
[—]Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611 USA

The advances in Deep Neural Network (DNN) architecture and computational capability of modern computers over the last decade, has led to extensive research and development in the field of character recognition and image classification. This study proposes a DNN architecture to effectively classify hand written characters over a data set created by combined effort of students and instructors of EEL-5840/4773 course. In our study, we compare the performance of Convolutional Neural Network (CNN) and Support Vector Machines (SVM) with a baseline *k*-Nearest Neighbors(*k*-NN) model. The metric to compare model performance is determined by the accuracy of classification and all models are tuned using hyperparameter tuning techniques to maximize performance of each model.

***Index Terms*—Machine Learning, Convolutional Neural Networks, Support Vector Machines, *k*NN, Data Classification.**

I. Introduction

OVER the years, machine learning has developed into a widely used framework for approximating statistical and algebraic relations over a given set of data. The ability of algorithms to accurately predict outcomes and generalize relationships between data sets has been beneficial in many working environments including health-care, government, transportation, and more. The two most widely known methods for machine learning are supervised learning and unsupervised learning. Supervised learning algorithms learn and train with labeled data. In this sense, the inputs are given to the model and the outputs are compared with target values. A few applications for supervised learning are regression, classification, and prediction. Unlike the aforementioned learning method, unsupervised learning does not learn and train with labeled data. Instead, it learns the structure and similarities within the unlabeled data. Popular applications for unsupervised learning are clustering, classification and function approximation.

Contributions: The contributions of this study are summarized as follows:

- To create a complete ML end-to-end pipeline for hand-written character recognition.
- We propose and compare performance of three different ways to classify our data sets implementing *k*-Nearest Neighbors, Support Vector Machines, and Convolutional Neural Networks. A performance comparison of each technique is made on the basis of accuracy of labels prediction and the computational efficiency of the algorithm.

II. Background/Literature Review

The challenging part about training machine learning models is properly tuning the hyper parameters for the technique of choice. We run into these very same challenges while implementing the *k*-Nearest Neighbors algorithm, Support Vector Machines with the RBF kernel, and Convolutional Neural Networks.

In the *k*-Nearest Neighbors algorithm, the hyper parameters to be tuned are the number of neighbors, number of weight vectors, and proper distance metric to produce the optimal result. The distance metric is usually given but Anava et al. [1] developed a novel approach that adaptively chooses the number of neighbors and associated weight vectors for each decision point by making the bias-variance trade-off explicit.

Support Vector Machines are vastly different from the *k*-Nearest Neighbors algorithm but still aim to classify data as accurately as possible. Pradhan [2] discusses the goal of a SVM: to find an optimal hyper plane that can be used to classify patterns in data that maximize the margins of the hyper plane. Maximizing the margins of the hyper plane leads to greater success in accuracy rate when classifying patterns in data.

The use of Convolutional Neural Networks (CNN) has been extensively researched in recent years. CNNs were first introduced for image classification [3]. Due to the feature extraction and selection ability of CNN, the CNNs provide robust results in 2D image classification and segmentation [4].The key idea in the working of convolutional layer is that the input sequence is mapped to a lower dimension space using operators (e.g., Soft-Max), so that only high-level features are selected. The CNN layer has proved to be extremely useful in data preprocessing for arbitrary lengths of sequence data due to its ability to extract high-level features. As the two aforementioned methods, CNNs hyper parameters of weights and activation functions must be fine tuned to get the optimal results. As beneficial as they are, certain situations as binarizing the network can trade accuracy loss for a reduction in memory size and ease of computation. Lin et al. [5] aimed to optimize this approach by approximating full precision weights with linear combinations of multiple binary weight bases along with the execution of multiple binary activation function to handle information losses. With the proper tuning, the resultant model will meet similar accuracy as a full precision CNN.

III. Implementation

A. Data Set

The data set considered contains 6720 samples of 10 classes of handwritten digits. The 10 classes are as follows: a, b, c, d, e, f, g, h, \$, and # with labels 0-9. The distribution of each class is given by the histogram in Fig 1.

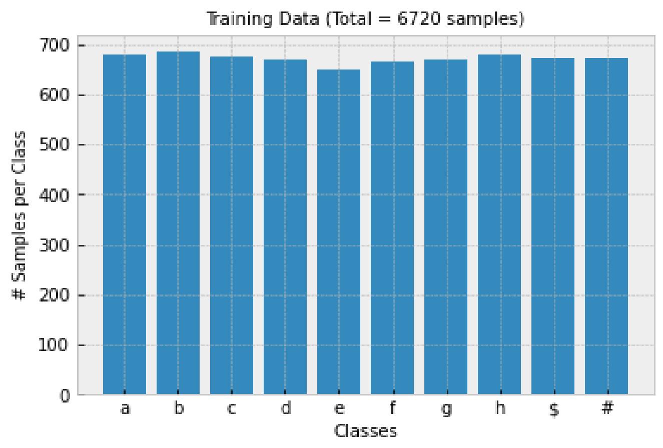

Fig. 1. Histogram of number of samples of each class

EMNIST Balanced Test Set was used to add capital letters since the original data had few such samples. Numbers and any capital letters which were not in the collected data set were assigned to a new class ('10') for detection of unknown samples in hard test set and numbers $3, 4, 6, 9$ were removed due to similarity with collected data.

B. Data Preprocessing and Augmentation

The preprocessing begins by correcting the mislabeled data. For this we manually inspect the data samples and then update the incorrect labels. OpenCV library [6] is used to resize and perform the transformations. We applied a Gaussian Blur with kernel size 3 to the images to remove noise. The data set was augmented with cropped, translated, and flipped images. Images were turned from gray scale to black and white using different thresholding techniques. These images were used to create bounding boxes using contour detection. Images were cropped using these bounding boxes. Cropped images were created for both the original gray scale image and the black & white image. More images were added by adding translated versions of the cropped and original images. Similarly, a horizontal flipped version of the image was also used. Therefore, for each sample, we created $3 \cdot (1\ threshold\ crop + 1\ original\ crop + 1\ original) \times 2 \cdot (flip) \times 2 \cdot (translate) = 12$ images. The data from EMNIST data was not processed since it is already a preprocessed dataset.

C. k-NN, SVM and CNN Implementation

We use the open source sklearn library [7] to implement *k*-NN and SVM algorithms. The Yellowbrick [8] library is used to visualize the results of the algorithms. The train and test split is considered to be 80% and 20%, respectively, for both algorithms. The data set is preprocessed using OpenCV to create contours, resize and normalize images. No data augmentation in terms of increasing sample size is performed for *k*-NN and SVM as the sample size is sufficient to train both algorithms.

For implementing CNN we use the open source tensorflow library [9]. The training and validation split is considered to be 80% and 20%, respectively. 20% validation split is considered on the training set to ensure generalization of the trained algorithm for unknown samples. Data augmentation is performed on the training set and additional samples from the EMNIST data set are added to the model, a detailed description is given in Section IV-A. All models are trained and tested using HiPerGator computing resources.

IV. Experiments

A. Data preprocessing and augmentation

Using 90000 features would bring Curse of Dimensionality to our model. Also, training a model with 300x300 pixel images would be both resource intensive and time consuming. The amount of data was also limited. Considering these factors, we decided to use images with smaller dimensions. We compared two image sizes- 28x28 and 50x50 for accuracy and chose 50x50 since it resulted in much better performance. Since a large part of the data set contained images with small characters, it was important to crop images before scaling down. A common way to find objects in data is creating bounding boxes using contour detection. A single method to find bounding boxes did not work well since there was a lot of variability and noise in the data. As a result, we experimented three methods- Binary Thresholding[10], Otsu's Method[11], and Canny Edge detection[12] and found that they contrast each other well so it would be useful to apply all these techniques and choose the best one for each sample.

Otsu's Method finds the best threshold value for each pixel using its surrounding pixels. It can detect faint characters easily but, images with patches or non-uniform images result in images with large patches. Ruled lines are also accentuated.

Binary Thresholding applies a common threshold to all pixels which does not form patches in non-uniform images but characters in images with light handwriting disappear after thresholding

Edge Detection: We use Canny Edge detection to detects edges. Hough Line Transform can be used to detect any straight lines in an image and since it works well after edge detection, we apply Hough line transforms to find and remove ruled lines. This method works well for images with many ruled lines but like binary threshold, it results in blank images for some samples

All these methods were used to find the largest contours in an image such that its enclosing circle remained within the image boundary(to ensure that large patches are not selected). The largest contour out of the three was chosen and was cropped. Fig 2 shows some of the original images and their respective processed versions

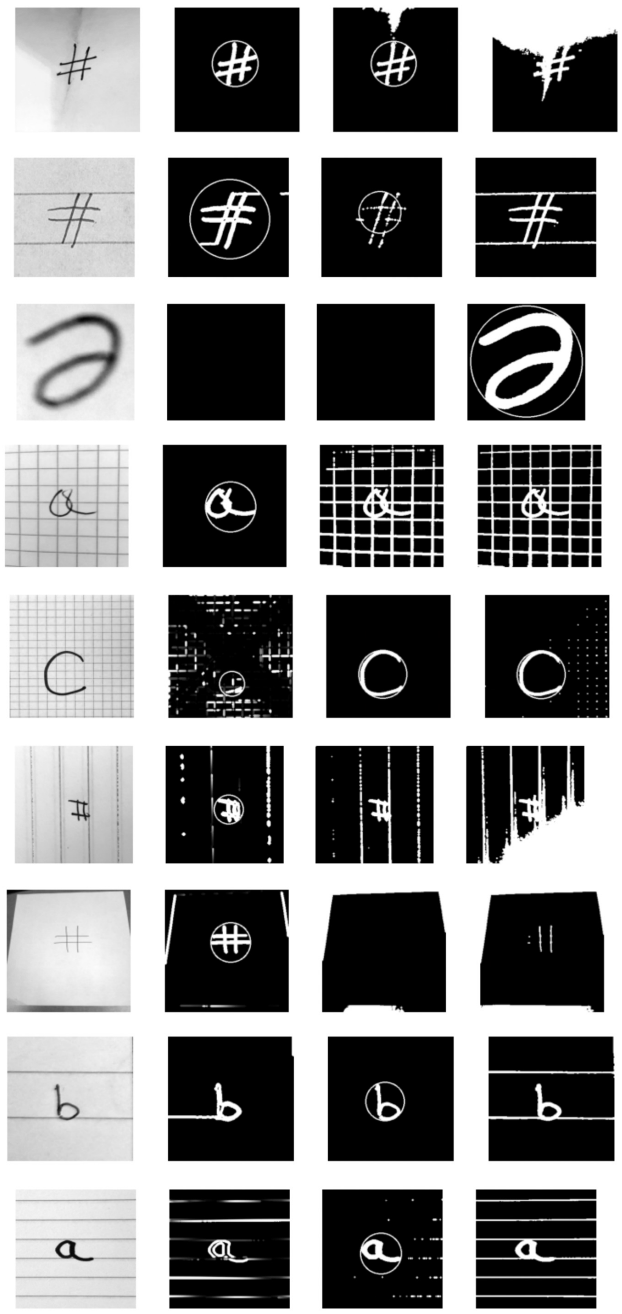

Fig. 2. 1. Original Image, and Images with enclosing circles for 2. Edge Detection Technique, 3. Binary Threshold, and 4. Otsu's Method from left to right

B. *Hyperparameter tuning for k-NN and SVM*

k-NN requires only hyperparameter to be tuned, namely, the number of neighbors k. the number of neighbors parameter denotes the number of closest neighbor classes which determine the class of the sample. The hyperparameter is selected using Cross Validation(CV) Grid search between the range $[2, 60]$. Similarly, SVM also requires only one hyperparameter to be tuned, coefficient C. C decides the width of the margin during classification, the margin decides whether is a point is allowed to be classified incorrectly. The hyperparameter C is selected using CV grid search for a range $[0.01, 100]$.

C. *Hyperparameter tuning for CNN*

The hyperparameter tuning for CNN is computationally expensive as there are a large number of hyperparameters. We follow the following procedure: (i) Number of Convolutional Layers to be considered; (ii) Number of Feature Maps in each layer; (iii) Number of Neurons in the dense layer before the output layer. Fig 3 denotes the number of convolutional layers selected based on running the algorithm for 20 epochs. We only consider 20 epochs to see if the model with selected layer performs relatively better. For each model an additional convolutional layer is added with 32 feature maps in Fig 3. We perform the same experiment for number of feature maps and the number of neurons in the dense layer. The choice for selection of parameters are made on the basis of computational complexity increase compared to the relative increase in performance. Performance is determined by comparing the validation accuracy. Fig 4 denotes the number of maps selected in each layer based on the validation accuracy.Fig 5 denotes the number of neurons selected in dense layers based on the validation accuracy.

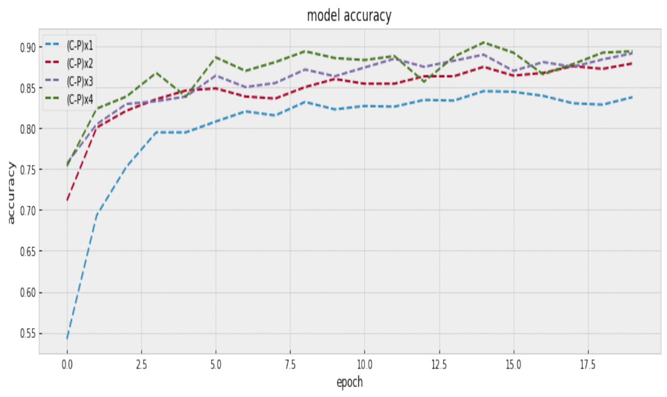

Fig. 3. Hyperparameter tuning for CNN: Selecting number of convolutional layers

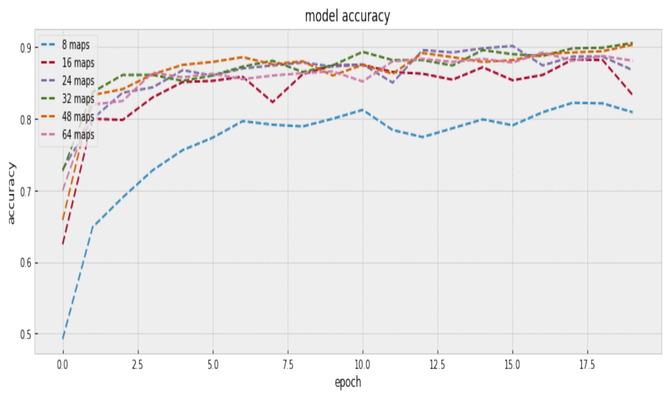

Fig. 4. Hyperparameter tuning for CNN: Selecting number of feature maps

The learning rate is considered to be 0.0001, the optimizer selected is 'Adam', batch size is considered as 64. These hyperparameters were selected based on testing the algorithm with keras tuner[13], we use the random search in keras tuner. Further details of the parameters are provided in section IV-D.

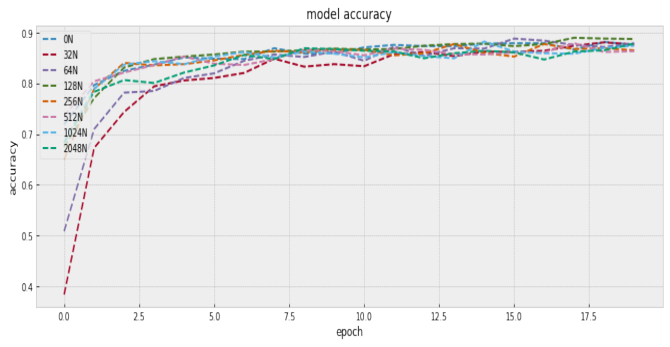

Fig. 5. Hyperparameter tuning for CNN: Selecting number of hidden units in dense layers

D. *Convolutional Neural Networks (CNN) architecture*

The CNN was inspired from the LeNet model[14]. It consists of three convolution-pooling layers and 3 dense layers. The convolution layers are arranged as follows- 16 5×5 filters, 32 5×5 filters, and 64 3×3 filters. Each layer is followed by a max-pool layer of size 2×2. The output from these layers is flattened and passed to the feedforward net with 3 units of size 256, 128, and 11 respectively. There is also a dropout layer between each of the dense layers. The CNN was also trained to recognize unknown classes. If any prediction was assigned to the label '10'or had a probability of prediction less than 60%, it was marked as unknown '-1' class. As seen from the accuracy plot 6, accuracy does not change significantly after 250 epochs. The model is trained for 256 epochs following this information to avoid any overfitting.

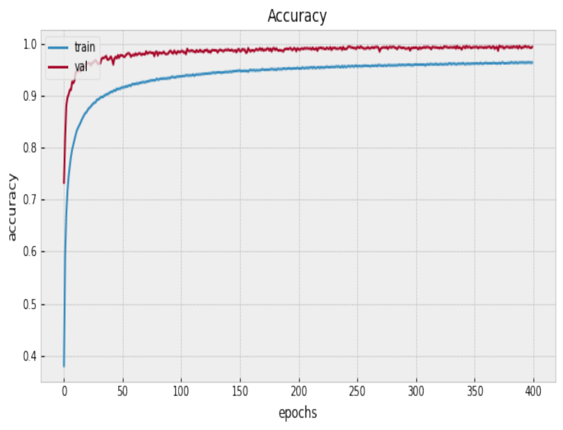

Fig. 6. Accuracy vs Number of epochs

V. Conclusion

Our experiments show that the CNN outperforms k-NN and SVM. The CNN test accuracy is 96% compared to 76% and 83% for k-NN and SVM, respectively. An important consideration is the computational load of training the CNN, which is significantly higher than training k-NN and SVM. k-NN requires the least amount of training time followed by SVM and CNN. Testing with various different configurations shows that high validation accuracy is not achieved by a single technique or model architecture but by combination of algorithm tuning and preprocessing of the data set available.

Acknowledgment

The authors would like to thank Dr. Catia Silva for providing this opportunity to experiment Machine Learning models and University of Florida's High Performance Computing for providing access to supercomputer GPUs.

References

[1] Oren Anava and Kfir Levy. k*-nearest neighbors: From global to local. *Advances in neural information processing systems*, 29, 2016.

[2] Ashis Pradhan. Support vector machine-a survey. *International Journal of Emerging Technology and Advanced Engineering*, 2(8):82–85, 2012.

[3] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[4] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[5] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *Advances in neural information processing systems*, 30, 2017.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] Benjamin Bengfort and Rebecca Bilbro. Yellowbrick.

[9] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] Payel Roy, Saurab Dutta, Nilanjan Dey, Goutami Dey, Sayan Chakraborty, and Ruben Ray. Adaptive thresholding: A comparative study. In *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pages 1182–1186, 2014.

[11] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[12] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[13] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. https://github.com/keras-team/keras-tuner, 2019.

[14] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.