Final Project – Handwritten Digits Classification

Kristine Antonyan, University of Florida, FL, kantonyan@ufl.edu
Tezcan Ozrazgat Baslanti, University of Florida, FL, tezcan@ufl.edu
Carter Kelly, University of Florida, FL, carter.kelly@ufl.edu
Hunter Smith, University of Florida, FL, huntersmith@ufl.edu

***Abstract* — The goal of this study is to achieve a good performance in classification of handwritten digits. We summarized experiments that involve data processing, feature extraction, and hyperparameter tuning for machine learning algorithms, implemented k-nearest neighbors, support vector machine, artificial neural networks, and convolution neural networks. Our convolutional neural network performed best achieving an accuracy of 0.89 in the test cohort.**

Keywords— hand digit classification, image processing, machine learning, kNN, SVM, CNN, neural networks.

I. Introduction

A. Character recognition problem research

The handwritten digits classification is part of the character recognition problem in Machine Learning. The two types of handwriting recognition techniques addressed in the literature are offline handwriting recognition (i.e., handwriting is implemented separately and scanned), and (b) online recognition (i.e. directly on the device). Among the many applications of handwritten character recognition are automatic classification of handwritten postal addresses, automatic processing of bank cheques, whiteboard reading, etc. [1, 2].

The research in character recognition problem tracks back into the 1960s and accelerates from the 1980s [3]. Among the two main issues addressed in the literature of developing a handwriting recognition system are the feature extraction and a classifier design [1]. Most studies identified in the literature have tested the publicly available CENPARMI, CEDAR, and MNIST databases.

B. Classifier design

The most frequently tested machine learning algorithms in the literature are Multilayer Perceptron, Support Vector machine (SVM), Naïve Bayes (NB), Bayes Net, Random Forest, J48, Random Tree, Logistic Regression (LR), and k-nearest neighbors (kNN). Several studies have reported a comparatively higher accuracy of the SVM algorithm [4, 5, 6]. Shirvastava & Gharde [5] reported the highest accuracy of 99.48%. Notably, the most recent literature examines the effectiveness of Neural Networks (NN) for handwritten digit recognition, such as deep neural network (DNN), deep belief network (DBN) and convolutional neural network (CNN) [7]. Ghoshet al. [8] revealed that DNA is the most effective, yet time costly NN algorithm (98.08% of accuracy).

C. Feature extraction

Due to the "curse of dimensionality", the higher dimensions create disadvantages such as overfitting, lesser interpretability and increase in training time. Hence, the studies of the last decade are consistently adopting strategies aimed at preserving the most information by projecting it into a lower space prior to the data classification (i.e., feature extraction). Das et al. [9] applied principal component analysis (PCA), linear discriminant analysis (LDA), and Isomap algorithms to MNIST dataset, followed by CNN classification and found out that PCA achieved the highest accuracy (99.29%), whereas LDA was the most efficient (98.34% accuracy, but only 9 dimensions with most information preserved and 75% reduced time consumption).

D. Current study

The current study implemented its own classification analysis with its own dataset of handwritten digits (0 to 9), aiming to reach the highest accuracy of the results. The reported best feature extraction techniques (i.e., PCA, LDA) followed by the most common classification algorithms (i.e., kNN, SVM, NB, LDA, LR, artificial neural network (ANN), CNN), were applied to the dataset.

II. Implementation

In order to accurately classify the provided set of handwritten digits, CNN was implemented in conjunction with a carefully optimized sequential series of data transformations. Using commonly available machine learning Python packages and libraries, each stage of this workflow was designed with the intent of not only maximizing the accuracy of the classification but ensuring that most personal computers would be able to replicate the process used in this study and its findings.

The general classification workflow was made up of two phases: data preprocessing and neural network training and validation. However, before either could be properly developed, the dataset itself had to be scrutinized to determine which methods, processes, and parameters would be most appropriate. The given dataset consisted of 3660 individual images, each a 300x300 pixel depiction of a digit from 0 to 9. Unlike other similar number classification datasets, there was little to no quality control exercised with the input data prior to its reception, so it was not uncommon for there to be lines, erroneous markings, rotated digits, or illegible (to humans) handwriting (as seen in Figure 1). As such, before any classification algorithm could be developed, a deliberate preprocessing methodology had to be designed and applied to the dataset in its entirety.

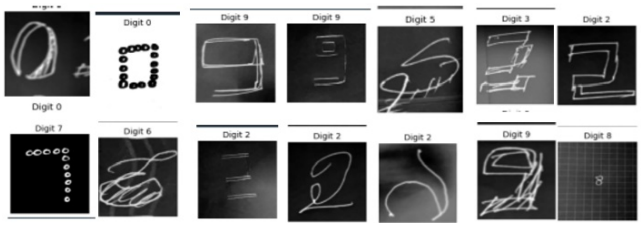

Figure 1. Assorted Examples of Provided Digits with Associated Labels

After importing the training data and associated labels, the data preprocessing workflow itself consisted of three separate morphological transformations (all of which used OpenCV's Image Processing tools): morphological gradient, resizing, and erosion. First, a morphological transformation (henceforth referred to as "outlining", cv2.MORPH_GRADIENT using a 60x60 kernel) was used to remove "noise" around the digits in the form of lines and erroneous markings, as well as to clearly define the markings that made up the edges of each digit. After outlining had been applied to all digits, each image was resized (cv2.resize()) from their original 300x300 pixel dimensions to 20x20 pixels to generalize pixel values (making it easier for the classifier to build profiles for each digit and distinguish between them) and compress the data for faster processing. Finally, each 20x20 digit underwent a morphological transformation (henceforth referred to as "outlining", cv2.MORPH_GRADIENT using a 1x1 kernel) which further defined the structure of each image and reduced the signature of any remaining noise. Application of all three preprocessing functions set the conditions for the CNN to train on (and eventually distinguish between) the dataset.

The second phase of the classification workflow was the CNN, which used the preprocessed dataset to train a model that was capable of distinguishing between the handwritten digits. The CNN itself was built using the Keras API (Application Programming Interface) and consisted of three sections: the input layer, multiple hidden layers (repetitions of convolutional, normalization, and pooling), and a classification layer. After importing the preprocessed data and its associated labels, the input layer reshaped the data from its 2-D array (number of samples, number of dimensions) to a 4-D array (number of samples, image width, image height, 1). It was then, through Keras' ImageDataGenerator, each individual image was randomly: rotated within a range of 15°, as well as zoomed in/out and expanded/contracted in width and height by a factor of range 0.2.

After completing the input layer, the dataset was fed into the hidden layers, which, as per Figure 2, consisted of multiple repetitions of convolutions and pooling. For this model, each convolution layer also incorporated a batch normalization and a relu activation (an example of which is shown in Figure 3) for a total of seven distinct convolutional layers. The filter sizes for each convolution ranged from 20 to 128, with kernel sizes ranging from a maximum of 5x5 pixels to a minim of 1x1 pixels. Through optimization, it was found that a batch size of 300 an epoch limit of 750 yielded the best results. Finally, the data was fed into the classification layer, which, using a two-layer 750 neuron MLP, assigned labels to each image. When tested on a limited dataset, this workflow provided an accuracy of 89.2%.

```
# Layer 1
model.add(Conv2D(20, (5, 5),
                 kernel_initializer='he_uniform',
                 input_shape=(img_rows,img_cols,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

Figure 3. Convolutional Layer Example (Filter Size 20, Kernel Size5x5) with Batch Normalization and Relu Activation

III. Experiments

The experiments conducted aimed at classifying the images of handwritten digits to their correct numerical identification. The dataset was built by the class, as each class member provided 10 sets of images of hand-written digits 0 to 9. The whole dataset was partitioned into a training set (70%, 3360 images) and a hold-out test set (30%, 1440 images). The images were uniformly vectorized to a 300x300 resolution and contained large variation, including shape, thickness, continuity, and background noise. A further normalization of training data was implemented followed by splitting that data into a 70% training and 30% testing datasets.

The experimental workflow (Figure 4) consisted of three main steps: data preprocessing, feature extraction, and

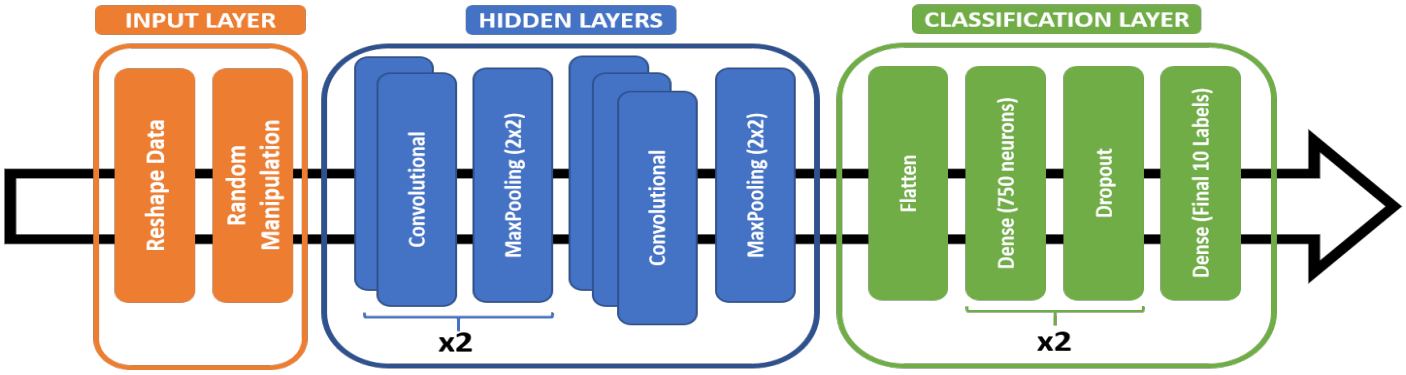

Figure 2: Architecture Used for CNN Model Generation and Training

classification methods. For each step, a variety of proven methods exist, and all combinations of these methods were tested to determine the workflow with highest classification accuracy. Randomization and replication were fulfilled throughout the experiment through the implementation of 5-fold cross validation.

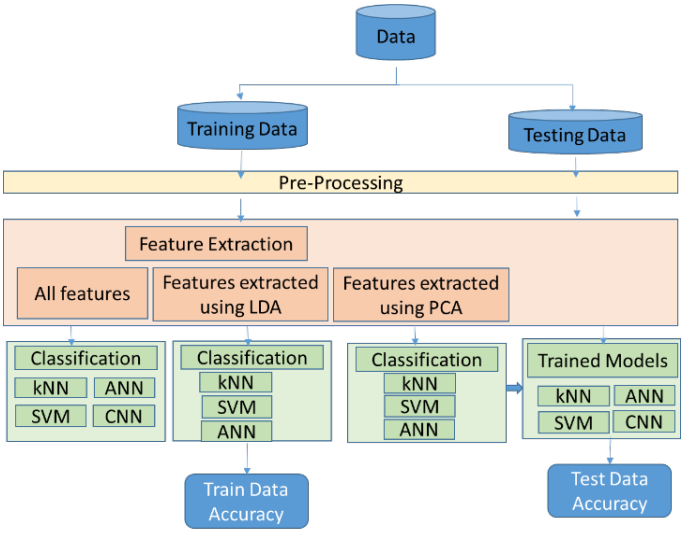

Figure 4. Implementation of hand-written digit classification

A. Data Preprocessing

To standardize and reduce noise in the data, a variety of preprocessing techniques were tested, including resizing image resolution and morphological transformations – opening, closing, dilation, and "outlining" or finding the morphological gradient. All combinations of these methods were tested for classification accuracy as the primary step in the workflow in a factorial design. The top performing permutation was determined to be 1) dilation, 2) resizing image resolution, and 3) "outlining" with selected hyperparameters: image resolution of 20x20, dilation kernel size of 60, and outlining kernel size of 4.

B. Feature Extraction

The second step of the workflow, feature extraction, included three methods: LDA, PCA, and no feature extraction. Each of these methods and the associated hyperparameter -- number of components -- were tested as the second step in the workflow, again in factorial design. This was a partial factorial design, as LDA was quickly eliminated from the workflow based on inferior accuracy to PCA. PCA improved accuracy to almost all workflows, the notable exception being those workflows implementing the CNN classifier. This was due to the limited dimensionality from PCA and LDA preventing the CNN from reshaping the vectorized digits into a grid large enough to apply its convolutions to. Table 1 displays the accuracies of selected workflows.

TABLE I.

TEST ACCURACIES OF BEST PERFORMING PREPROCESSING AND FEATURE EXTRACTION WORKFLOWS

Workflow Accuracy	Preprocessing Methods					
	Opened Only	***Outline Only***	***Thinning Only***	***Outline Only (v2)***	***Outline, Resize, Thin***	***Thin, Resize, Outline***
SVM – all features	0.38	0.57	0.58	0.6	0.56	0.58
SVM – LDA	0.42	0.6	0.6	0.63	0.56	0.58
SVM – PCA	0.42	0.6	0.6	0.63	0.56	0.58
kNN – all features	0.51	0.65	0.62	0.62	0.62	0.65
kNN – LDA	0.28	0.41	0.43	0.25	0.41	0.46
kNN – PCA	0.52	0.6	0.6	0.6	0.6	0.66
ANN – PCA	0.58	0.61	0.57	0.63	0.68	0.72
CNN – all features	0.75	0.81	0.82	0.85	0.89	0.88

C. Classification Algorithms

The final step of the experimental workflow was the classification. Six methods were tested including: NB, kNN, SVM with RBF kernel, LR, ANN, and CNN. We used L2-regularization for LDA and LR. LDA was evaluated using repeated stratified 5-fold cross-validation with 3 repeats. LR was evaluated using C values of 0.001,0.01,0.1,1,2.5, and 5, and maximum iteration of 1500. SVM was evaluated using RBF kernel with gamma values of 1e-3 and 1e-4 and C values of 1500, 2000, and 2500. Best selected parameters for the ANN were 2 neural layers with 300 neurons, 4000 epochs, and an adaptive learning rate initiated at 0.08. kNN model determined optimum number of neighbors to be 3. Of these, kNN, SVM, and ANN were tested in full factorial experimentation, and NB and LD were quickly eliminated from the experiment due to inferior accuracies. As previously noted, LDA and PCA feature extraction conflicted with CNN implementation, therefore limiting CNN to a partial factorial workflow. CNN was the best performing classifier, achieving an accuracy of 0.89 (Figure 5). The final workflow included no feature extraction, and selected CNN hyperparameters were a batch size of 300, 750 epochs, and neural network hidden layers of 750 neurons each.

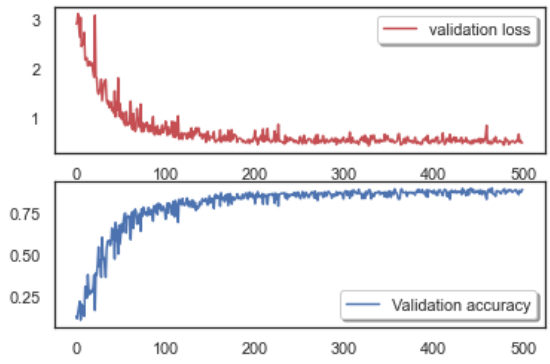

Figure 5. Validation Loss and Validation Accuracy of Final Model.

D. Final Results

The final workflow, from preprocessing through label assignment, produced a trained model capable of assigning labels to new data. Evaluation of this model using a 70/30 split of the provided 3660 data samples yielded an overall accuracy of 89.2% with confusion matrix as presented in Figure 6 and an AUC of 0.94 (Figure 7).

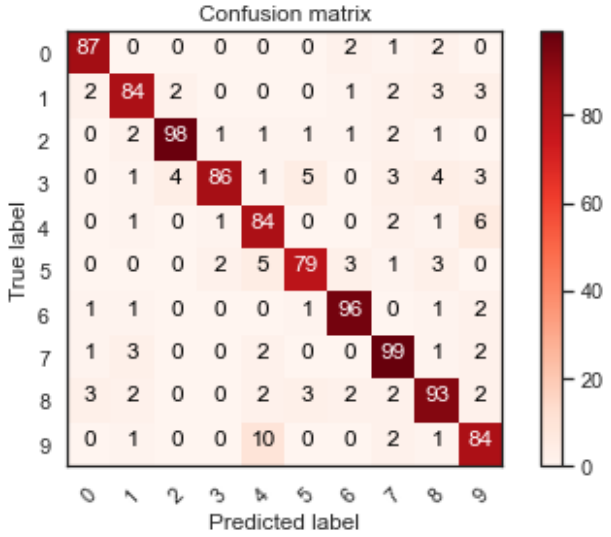

Figure 6. Confusion Matrix of Final Workflow with 89.2% Accuracy

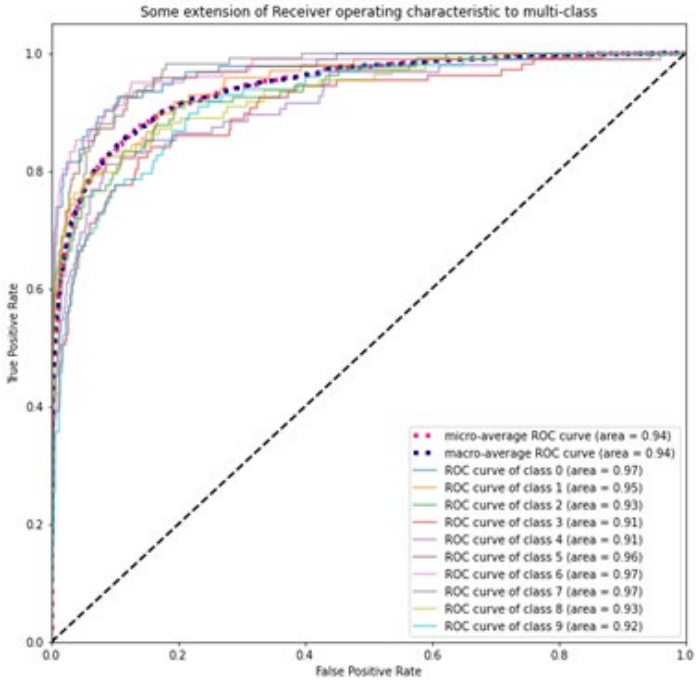

Figure 7: Receiver Operating Characteristics Curve

IV. Conclusions

Data preprocessing and feature extraction steps are crucial for improvement of accuracy for hand-written digit classification. Automatization of identification of best parameters would improve efficiency of algorithms significantly. We have selected data preprocessing and feature extraction methodology that works best for each machine learning algorithm used which include kNN, SVM, ANN, and CNN. CNN performed best achieving an accuracy of 0.89 in the test cohort. Obviously, our findings show less accuracy than others' results. Among some methodological differences, we believe, that the poor quality of our data has significantly affected our results, compared to other studies conducted with public datasets.

References

[1] Chacko, V. Krishnan, G. Raju, and P.B. Anto, "Handwritten character recognition using wavelet energy and extreme learning machine", International Journal of Machine Learning and Cybernetics, 3(2), 2012, pp. 149-161.

[2] T. Plötz, and G.A. Fink, "Markov models for offline handwriting recognition: a survey", International Journal on Document Analysis and Recognition (IJDAR), 12(4), 2009, p. 269.

[3] Purohit, and S. S. Chauhan, "A literature survey on handwritten character recognition", (IJCSIT) International Journal of Computer Science and Information Technologies 7, no. 1, 2016, pp. 1-5.

[4] S. M. Shamim, M. B. Miah, M. R. Angona Sarker, and A. Jobair, "Handwritten digit recognition using machine learning algorithms", Global Journal Of Computer Science And Technology, 2018.

[5] S. Shailedra Kumar, and S. S. Gharde. "Support vector machine for handwritten Devanagari numeral recognition", International journal of computer applications 7, no. 11, 2010, pp. 9-14.

[6] L. Cheng-Lin, K. Nakashima, H. Sako, and H. Fujisawa. "Handwritten digit recognition: benchmarking of state-of-the-art techniques" Pattern recognition 36, no. 10, 2003, pp. 2271-2285.

[7] Shrivastava, A. I. Jaggi, Sh. Gupta, and D. Gupta. "Handwritten Digit Recognition Using Machine Learning: A Review", 2019 2nd International Conference on Power Energy, Environment and Intelligent Control (PEEIC), IEEE, 2019, pp. 322-326.

[8] M. M. Ghosh, and A. Y. Maghari. "A comparative study on handwriting digit recognition using neural networks", 2017 International Conference on Promising Electronic Technologies (ICPET), IEEE, 2017, pp. 77-81.

[9] Das, K. Tuhin, and Ch. Saravanan. "Dimensionality reduction for handwritten digit recognition." EAI Endorsed Transactions on Cloud Systems 4, no. 13, 2018.

[10] S. Ahlawat, A. Choudhary, A. Nayyar, S. Singh, and B. Yoon. "Improved handwritten digit recognition using convolutional neural networks (CNN)." Sensors 20, no. 12, 2020, p. 3344.