Handwritten Optical Character Recognition with KNN and MLP Classifiers

Spencer Chang, Kaiyang Han, Fuyuan Zhang, Zhewei Song
{chang.spencer, khan2, fuyuanzhang, zhewei.song}@ufl.edu

***Abstract*—Recognizing English chracters written by multiple people is a tedious job for many workers, from mail sorters to document readers. Optical Character Recognition (OCR) uses technology, such as machine learning, to recognize different characters within a single document and turn it into a digital format. In this work, using two data sets of lower case handwritten English characters, K-Nearest Neighbor (KNN) and Multilayer Perceptron (MLP) classifiers are trained to do OCR. We use K-fold cross validation to test and verify chosen hyperparameter settings in the KNN and MLP models. On their respective data sets, the KNN achieved an average validation accuracy of 97.25%, and the MLP achieved an average of 74.86%.**

I. Introduction

Handwritten characters are typically simple objects for humans to recognize, but when it comes time to automate the process using learning algorithms, the problem becomes more difficult. Humans rely on many features to tell letters apart, but the difficulty for learning algorithms is what features are the best to use for differentiating letters at comparable performance.

In this work, we create one classifier each for two provided binary image data sets, one set containing only the letters 'a' and 'b' and the other containing the letters 'a','b', 'c', 'd', 'h', 'i', 'j', and'k'. There is overlap between the two data sets through the letters 'a' and 'b', but each of the following presented learning models are trained on only one of the data sets at a time. The given data sets has close to 800 samples per class label.

At the start of the learning machine pipeline (Figure 1), we perform image pre-processing to center each sample's bounding box of nonzero pixels in a square of size 50x50. Next, feature extraction makes use of many properties given by the Python programming library sci-kit image, specifically `regionprops`. The particular properties chosen are covered in Section II.

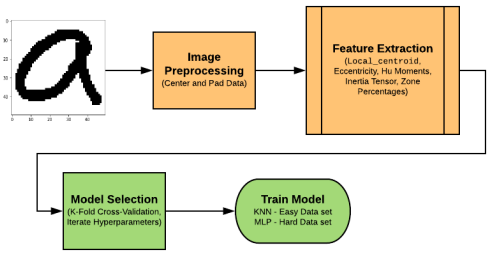

Fig. 1. Pipeline of Learning System

The features are then input for model selection. We chose K-Nearest Neighbor and a simple neural network to use for the two-class data set and for the eight-class data set, respectively. We then performed K-Fold Cross-validation for each data set and model pair to verify their hyperparameters and effectiveness. Finally, both models are trained on their respective training sets to prepare them for the testing set to be revealed at a later date. Such a data set was not yet provided to the group to prevent unintentional information leaks.

In Section II, our implementation is described, including feature extraction and model implementations. Section III details the experimental validation used to verify the best model hyperparameters. Finally, conclusions are drawn about this study in Section IV.

A. Literature Review

Using the ideas of joint random variables in 2D, characteristic functions, and moment-generating functions, Hu developed a general theorem for generating absolute orthogonal moment invariants for pattern recognition in images [1]. Specifically, the geometric moments are meant to be translation, scale, and rotation invariant features. Other researchers have listed between 32 and 52 Hu invariants generated using his theorem [2]. Later, Reiss has shown the Hu invariants to be wrong but also corrected them [3]. Another extracted feature would be moment of inertia, which has been used recently for classifying handwritten Arabic digits [4].

Prior research for optical character recognition covers many different classifier approaches [5]. In this work, the focus is on the K-Nearest Neighbor (KNN) and Multilayer Perceptron (MLP) classifiers. KNN classifiers have been used for Arabic and Gujarati [6], [7] or for custom classifiers [8], [9]. For MLP approaches, studies have applied the network to multiple languages and types of characters, such as Arabic [10] and handwritten English [11].

II. Implementation

In this section, we describe the pipeline for our learning system. In particular, this section covers the image preprocessing, feature extraction, and model implementations. Additionally, the following Python programming libraries were crucial for our implementation: NumPy, Sci-kit Image, and PyTorch.

A. Image Preprocessing

The binary image training data sometimes came with handwritten characters in the corners with white space filling

the rest of the image space. To resolve most of the issues, a bounding box was calculated for nonzero pixel locations within a provided sample. After creating the bounding box, the image is re-padded with zeros such that the bounding box's center is now the center of a 50x50 binary image. Most training samples that were viewed had bounding boxes less than the square size of 50x50, and it was found that this technique successfully processed the images to be size 50x50 with most, if not all, characters at the center of the images.

Using the Sci-kit Image library, each image is then processed in the order of `closing`, `remove_small_objects`, and `skeletonize`. We used a diamond-shaped neighborhood with height and width of 5 and sub-image windows of size 6x6 to close gaps between parts of images, in case characters came out segmented or incomplete. Components of size less than 10 pixels were filtered out of sample images. Finally, to remove the influence of stroke thickness, the character's skeleton is computed.

B. Feature Extraction

From the sample's character skeleton, four sets of features are generated using Sci-kit Image's `regionprops` method, with the total vector length equal to 22 dimensions, seen in Table I. `regionprops` will return a number of properties per each identified region. In binary images, there should only be one region as the only difference between the image background and the foreground is whether the pixel is zero or nonzero, respectively. The first feature is the local centroid of the image, defined as the centroid of the bounding box of the image. That is, it will calculate the center of mass only for a bounding box for one region. Second, we extract the eccentricity of the region, which considers the region's enclosed pixels as an elliptical shape and calculates the ratio of the focal distance (the region's edges also count as enclosing the pixels). Eccentricity is helpful for determining how wide or tall a character might be. If used to compare an 'i' to an 'a', the focal distance for the former should be greater than the latter.

The third set of features is a set of seven Hu moments [1], [3]. By considering moments, we are treating the image as a probability density function of a random variable. Such calculated 2D moments consider the case for two joint random variables X and Y. Derivation details for the Hu's general theorem of moments can be found in his and Reiss's work [1], [3]. Overall, the generated moments are meant to provide translation, rotation, and scale invariant features.

The second to last set of features is an inertia tensor of moments of inertia, a common concept in statics and dynamics of structures and physical bodies. The moment of inertia is an object's tendency to resist angular acceleration about some axis. In regard to the samples, the moment of inertia considers the horizontal (X) and vertical (Y) axes centered at the center of mass as lines which the region will rotate around. The tensor

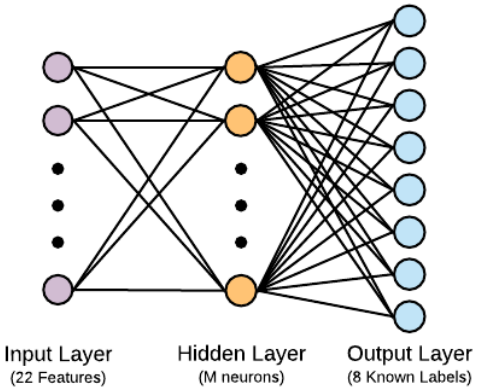

Fig. 2. Implementation of MLP with M = 16

contains the following values:

$$\begin{bmatrix} I_{XX} & I_{XY} \\ I_{YX} & I_{YY} \end{bmatrix}$$

I_{XX} is the moment of inertia around the centered X-axis, while the I_{YY} is the moment of inertia around the centered Y-axis. I_{XY} and I_{YX} are the moments of inertia around a diagonal line $Y = X$ through the center of the region (ie. $I_{XY} = I_{YX}$). The intuition for using these comes from considering the inertial differences between the letters 'a' and 'b'. Rotating about the Y-axis, both might hold similar values, but when rotating around the X-axis, we would expect I_{XX} for 'b' to be greater than that of 'a,' meaning that 'b' would 'resist' more angular acceleartion about the X-axis than 'a.' A higher proportion of b's mass is further away from the center of mass in the Y-direction than a's, resulting in this higher resistance to angular acceleration.

To get the final set of features, each sample binary image is split into a grid of 3x3 squares, each size 16x16. We then take the sum of the nonzero pixels in each square, divide by the sum of all nonzero pixels and append them to the end of the feature list.

TABLE I
FEATURE VECTOR VALUES

Feature #	Type
1	X-Coordinate of `local_centroid`
2	Y-Coordinate of `local_centroid`
3	Eccentricity
4-10	Hu Moments 1-7
11-13	Inertia Tensor Values (I_{XX}, I_{XY}, I_{YY})
14-22	Zone Percentages for 3x3 Grid

C. Learning Models

The first model used is a K-Nearest Neighbor (KNN) model [12]. Using the training data as a set of clustered classes, the KNN classifier labels new samples based on their proximity to the K closest neighbors. The label by majority vote of the closest neighbors. That is, if most of the K neighbors are

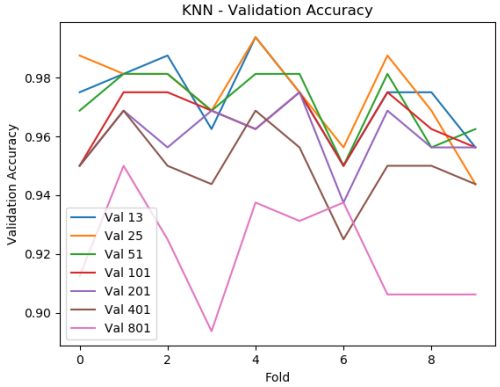

Fig. 3. KNN with Neighbor Values of 13, 25, 51, 101, 201, 401, and 801

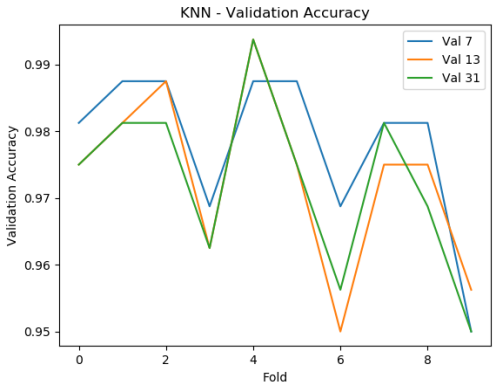

Fig. 4. KNN with Neighbor Values of 7, 13, and 31

in class 1, the new sample is then labeled as class 1. We implemented KNN for the easy data set using the following hyperparameters: K = 31, Minkowski distance metric, and weighted distances. These choices are discussed in Section III.

The second model, a Multi-Layer Perceptron (MLP) model [13], is used for the hard data set. The MLP is created from fully-connected layers, where there is one input layer, one or more hidden layers, and one output layer. Each neuron takes the outputs from a previous layers neurons as a weighted sum plus a bias value, which is then fed into a nonlinear activation function that outputs to the next layer's neurons. To train the MLP model, we use backpropagation to update weights, allowing the model to learn decision boundaries between classes. Our MLP uses the sigmoid activation function for all neurons, the Adam optimizer for backpropagation, and a single hidden layer with $M = 16$, as seen in Figure 2, with a learning rate of 0.01.

III. Experiments

In this section, we detail the experiments used to check model hyperparameters and which configurations worked best for each data set. As a reminder, KNN is used for the easy data set, and the MLP is used for the hard data set.

A. KNN - Choosing K Neighbors

To decide the optimal number of K neighbors for the KNN classifier, we did K-Fold cross-validation with 10 folds. Fewer folds would make it harder to see variance between folds as the classifier does not train over a number of epochs as the MLP. To test, the random state of shuffling K folds was fixed and the number of neighbors was varied in two phases. In the first phase, neighbor values of 13, 25, 51, 101, 201, 401, and 801 were tested, as seen in Figure 3. From the first phase, it was found that factoring in more neighbors (ie. 101, 201, 401, and 801) was detrimental to performance. After searching lower values, the 31 was decided due to its slightly lower variance across folds and speed of inference, as seen by the validation accuracy in Figure 4. Across ten folds, the average validation accuracy with K = 31 is approximately 97.25%.

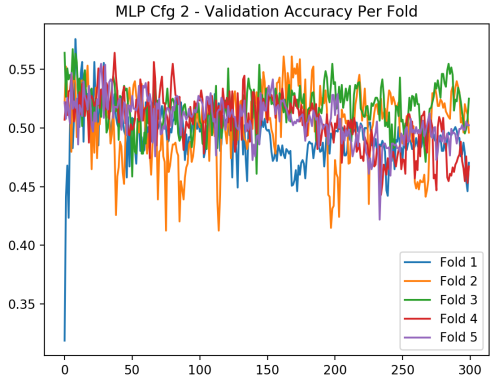

Fig. 5. Validation Accuracy from Using Two Hidden Layers in MLP (Validation Accuracy Vs. Epochs)

B. MLP - Number of Hidden Layers

Using 5-fold cross-validation, three main tests were conducted to determine the number of layers, number of neurons in layers, and the learning rate for the MLP. Given that we change one hyperparameter at a time and keep all others constant, an optimal configuration is not expected. The Universal Approximation Theorem (UAT) tell us that 1 is a sufficient number of hidden layers in an MLP to approximate the mapping from input to output [13]. However, it is not given to be optimal. Thus, the first test held constant the number of epochs at 300, the learning rate at 0.2, and the number of neurons in each layer, where the per-layer configurations are [22, 18, 12, 8] (three hidden), [18, 12, 8] (two hidden), and [12, 8] (one hidden). We chose to cap the number of neurons in the first layer to be less than or equal to the number of features. Results for two hidden layers can be seen in Figure 5. Too much variance occurred across folds with two hidden layers, and it appeared worse still for the three hidden layers. Thus, we stick with one hidden layer of neurons in the next two tests, considering it the optimal choice particular to this problem of OCR.

C. MLP - Learning Rate

The per-layer configuration is held constant at [12, 8] and the number of epochs at 300. We iterated over learning rates

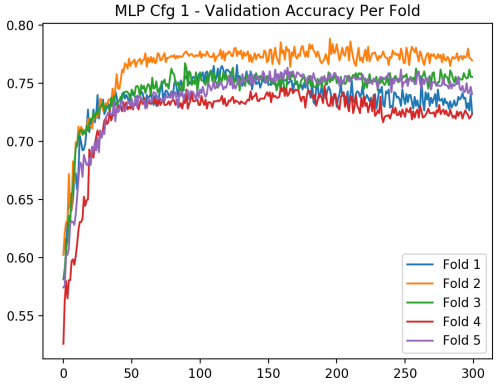

Fig. 6. Validation Accuracy for 20 Neurons in Hidden Layer and a Learning Rate of 0.01. (Validation Accuracy Vs. Epochs)

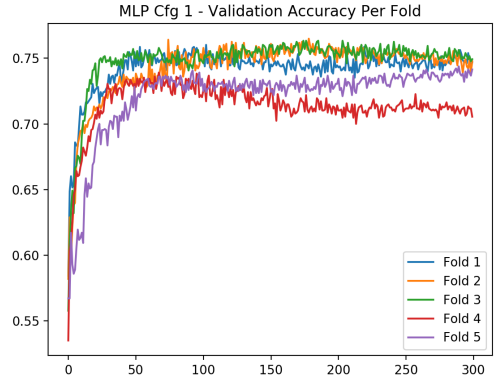

Fig. 7. Validation Accuracy for 16 Neurons in Hidden Layer and a Learning Rate of 0.01. (Validation Accuracy Vs. Epochs)

of [0.2, 0.02, and 0.002]. The first showed too much variance, where the validation accuracy bounced around much more than using learning rates of 0.02 or 0.002. It was also observed that a learning rate of 0.02 caused the MLP to overfit slightly toward the end of training. Moving on from here, the learning rate was decreased to 0.01 instead.

D. MLP - Number of Neurons

Given that we are using one hidden layer, we now have a trade-off between good approximation of the training data and generalizing from the training data to test data not yet seen [13]. If the number of neurons is too large, it tends to memorize the training data too well, instead of generalizing to new samples.

From subsection III-B, we hold the number of hidden layers at 1. Hence, in this last experiment, we iterate over another three neuron configurations: [20, 8], [12, 8], and [4, 8]. The learning rate is held constant at 0.01 and the number of epochs at 300. It was observed that the best validation accuracy may have occurred between the first two configurations. Thus, a configuration of [16, 8] was also used. Results from a configuration of [20, 8] and [16, 8] are seen in Figures 6 and 7, respectively. Overfitting appears to be present later in training, so the number of epochs is dropped to 200 for the final trained model. The average cross-validation accuracy was 74.86%.

IV. Conclusions

The experiments demonstrate that for the easy data, KNN gave a satisfying result while being faster than MLP. For the hard data, MLP offered a better classifying result. For KNN, we saw that considering too many neighbors will lower the accuracy. By searching semi-exhaustively, 31 was the final decision, balancing the ratio between speed and variance. For MLP, to get an optimal structure, we considered three hyperparameters: number of hidden layers, learning rate, and number of neurons. For this model, our experiments showed that a single hidden layer MLP gave the best performance with the least variance. The learning rate of 0.01 displayed less variance during cross-validation. For the number of neurons, validation showed that between 20 and 12 may give out the best accuracy, so 16 was chosen to balance the relationship between generalization and training data accuracy.

References

[1] Ming-Kuei Hu, "Visual pattern recognition by moment invariants," *IRE Transactions on Information Theory*, vol. 8, no. 2, February 1962.

[2] Øivind Due Trier, A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition-a survey," *Pattern Recognition*, vol. 29, no. 4, pp. 641 – 662, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0031320395001182

[3] T. H. Reiss, "The revised fundamental theorem of moment invariants," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 830–834, Aug 1991.

[4] B. K. Prasad, "Moment of inertia-based approach to recognize arabic handwritten numerals," in *Innovations in Electronics and Communication Engineering*, H. S. Saini, R. K. Singh, G. Kumar, G. Rather, and K. Santhi, Eds. Singapore: Springer Singapore, 2019, pp. 253–260.

[5] C. Y. Suen, M. Berthod, and S. Mori, "Automatic recognition of handprinted characters—the state of the art," *Proceedings of the IEEE*, vol. 68, no. 4, pp. 469–487, April 1980.

[6] J. H. AlKhateeb, F. Khelifi, J. Jiang, and S. S. Ipson, "A new approach for off-line handwritten arabic word recognition using knn classifier," in *2009 IEEE International Conference on Signal and Image Processing Applications*, Nov 2009, pp. 191–194.

[7] S. A. Chaudhari and R. M. Gulati, "An ocr for separation and identification of mixed english — gujarati digits using knn classifier," in *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)*, March 2013, pp. 190–193.

[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] M. Alabbas, R. S. Khudeyer, and S. Jaf, "Improved arabic characters recognition by combining multiple machine learning classifiers," in *2016 International Conference on Asian Language Processing (IALP)*, Nov 2016, pp. 262–265.

[10] A. Amin and W. Mansoor, "Recognition of printed arabic text using neural networks," in *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, vol. 2, Aug 1997, pp. 612–615 vol.2.

[11] M. Sabourin and A. Mitiche, "Optical character recognition by a neural network," *Neural Networks*, vol. 5, no. 5, pp. 843 – 852, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005801443

[12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[13] S. O. Haykin, *Neural Networks and Learning Machines*. London, England: Pearson Education Inc, 2009.