Project 01: CNN and KNN Approaches to Letter Classification

Ross Spencer, Yutai Zhou, Caleb Robey

***Abstract*—This project's aim is to perform character recognition. Given binary images containing one character each, we have taken 2 approaches toward classifying the picture into one of 3 or 9 categories, depending on the dataset. This paper leverages a convolutional neural network and KNN with feature extraction to attempt to find an accurate method, with 2 further implementations of a vanilla pixel-based KNN and a traditional vector neural net (taking each pixel as an input) without a convolution layer. Inspired by the robust power of CNNs in similar applications, we find that our CNN outperforms our KNNs and traditional NN on real-world data.**

***Index Terms*—Computer Vision, Machine Learning, Statistical Learning, KNN, Neural Network, Convolutional Neural Network**

I. Introduction

How do we perform character recognition on messy real-world scans of handwriting? In this project, we present 2 approaches to determine which character is contained within a binary image.

Based off of papers on achieving high-accuracy character recognition, we chose to evaluate using a K-nearest neighbors classifier and a convolutional neural network based off of T. Makkar, 2017, showing that both achieved over 96% accuracy on the MNIST dataset for digit recognition.

Since we chose to use 2 convolution layers in our CNN, we were curious to see how similarly implementing feature extraction in a KNN would compare to a KNN using solely the raw pixel data of each image stretched out into 1 long line. Basing our features off of those described by Frey et al. in 1991, we chose to recreate a more modern approach to their paper, using KNN instead of a Holland-style learning classifier system.

II. Implementation

Our first implementation relies upon a K-Nearest Neighbors classifier, utilizing feature extraction following features described by Frey et al., 1991, for use in a different classifier. The features include the x-axis center and y-axis center of the bounding box, the height and width of the bounding box, the total number of on-pixels in the zero-padded image, the coordinates of the centroid of the letter, the average distances from the centroid across both the x and y axes, the correlation of the horizontal variance with the vertical position, the correlation of the vertical variance with the horizontal position, the average number of horizontal edges in cross sections along the y-axis, the sum of the vertical locations of the aforementioned horizontal edges, the average number of vertical edges in cross sections along the x-axis, and the sum of the horizontal locations the aforementioned vertical edges.

To extract these features, we first pad by 0s around to make each image a uniform size, then in a for-loop call methods that extract each of these features. To scale each feature so that no particular feature dominates the others, we scale between 0 and 15 as per the methodology of Frey et al., 1991. After that, we feed the 6192 by 16 array into the KNN classifier along with the correct label to train it. For comparison's sake, we also implement a KNN using the raw pixel values to see how each performed, highlights of which are in the experiment section. Based on plots of the overall accuracy across all 9 classes and the micro and macro precision when treating all non-'A or B's as an 'other' class, we find that using 1 nearest

This work was supported in part by Publix cookies and fruity candy canes.

R. Spencer, Y. Zhou, and C. Robey are with the Machine Learning & Sensing Laboratory, and the Computer Science, Mathematics, and Electrical & Computer Engineering Departments at the University of Florida.

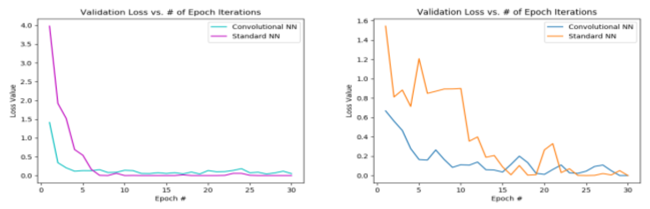
neighbor yields the best performance, replicated in our testing across multiple random states.

We utilized the PyTorch library to build our CNN and regular NN. We discuss the CNN in most detail because it was the most successful implementation. The CNN exists in a sequence of seven layers:

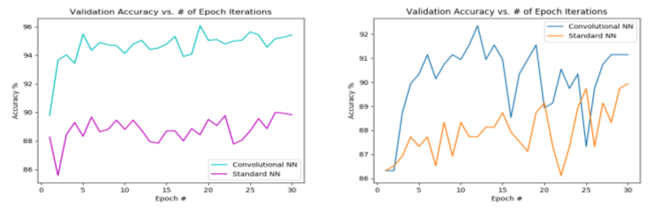
1. This is a convolution layer with one input channel (because the color space is binary, rather than RGB) and 30 output channels, which are feature maps to be processed in the next layer. This 2D layer applies a 5x5 kernel to execute the convolution and the activation function is ReLU.
2. The next layer is a max-pooling layer. This layer takes the 2x2 kernel max of the data. This should technically extract only the most crucial information from the images.
3. Layer 3 is also a 2D convolution layer with a ReLU activation function, but instead has a kernel size of 3x3.
4. Layer 4 is another max-pooling layer over a 2x2 kernel.
5. Now that the size of the data has been reduced by convolutions and max-pooling, layer 5 flattens the data so that it can be processed by a fully connected layer. This layer takes the data from a size of 7650 to 128.
6. Those 128 points are then selected down to 50 data points through the ReLU activation function.
7. Then there is one final fully connected layer that outputs without any activation function to the selected amount of output classes.

It is worth noting that we attempted to use the softmax activation function for the output and got accuracy results below 60%. The reason for that is still unclear.

The regular neural net was simply two layers. It had an input layer with the number of pixels in the images with the ReLU activation function. The output of that layer (500 output nodes) was then processed by a hidden layer that had an output quantity corresponding to the number of classes.



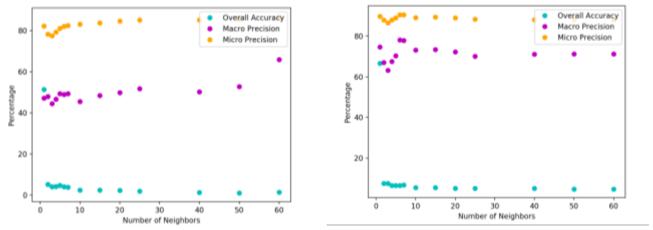
These plot depicts the loss at each epoch, note that the CNN's loss decreases quicker, decreasing the overall training time. Class synthetic data to the left and real-world data to the right.



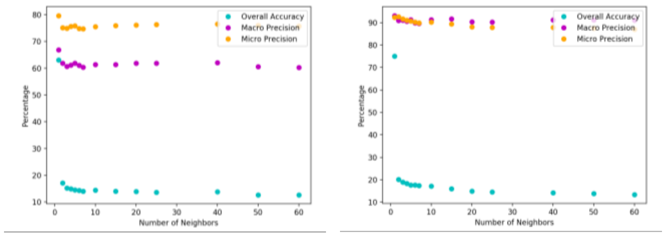
These plot depicts the accuracy of the model tested against the validation data at each epoch. Note that in both cases, the CNN's accuracy remains above that of the standard neural net. Class synthetic data to the left and real-world data to the right.

III. Experiments

To first decide how many neighbors to use in our KNN classifier implementation, we first looked at how the accuracy and precision changed as the number of neighbors varied. After generating plots for several random states, it is clear that both KNN implementations do best when classifying based off of only 1 nearest neighbor. Thus, the rest of our tests use only 1 nearest neighbor in their calculations.



These plots show the overall accuracy across all 9 classes and the macro and micro precision across classes 1, 2, and 'other,' with the left plotting the results of our features-based KNN and the right plotting those of our purely pixel-based KNN for comparison's sake on the original real-world dataset.



These plots again show the accuracy and precisions with our features-based KNN on the left and our purely pixel-based KNN on the right on our synthetic class-generated dataset.

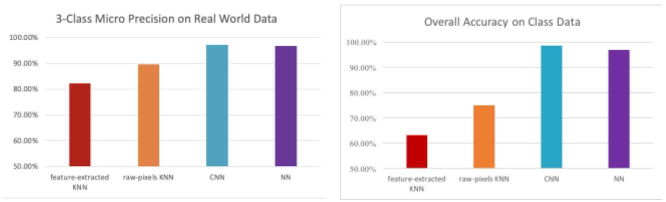
Further, while the micro precision and overall accuracy are around the same for both (with accuracy typically falling around ~10% less for the feature extracted KNN with 1 nearest neighbor), the macro precision takes a significant hit when classifying only off of the extracted features. However, running these iterations on our computers the raw pixel-based KNN took upwards of 8 minutes to generate the plot (14 iterations), whereas the feature-extracted KNN consistently finished in under a minute.

Overall, our macro precision on the original dataset shows that our features-based KNN performs relatively poorly on the original messy real-world dataset, slightly better than random guessing (50% macro precision as opposed to 33%).

On the other hand, both our NN and CNN perform very well on both datasets, with each reaching above 90% accuracy. CNN manages to edge out the NN, with validation accuracy at 97.34% on the real-world 3-class data and 98.53% on the synthetic class data across all 8 classes.

Between all 4 implementations of a character classifier, the performance between the 4 once all the design parameters were settled shows our CNN outperforming the other methods. The CNN also takes significantly less time than the NN to train, and, unlike the KNNs, doesn't have to reinvent the wheel to classify a new point.

To get the precision and accuracy, we ran the entire class synthetic dataset and the real-world dataset through the NN and CNN models. These were then checked against the corresponding labels.



These plots show the overall accuracy across all 9 classes and the macro and micro precision across classes 1, 2, and 'other,' with the left plotting the results of our features-based KNN and the right plotting those of our purely pixel-based KNN for comparison's sake on the original real-world dataset.

IV. Conclusions

The CNN appears to be the most successful model, probably mostly because it was designed specifically for image classification. The use of kernel spaces as inputs allows the neural net to learn not only a set of pixels, but rather information about kernels of pixels. This introduces a notion of locality to the analyzed feature space that is advantageous for image classification, because humans also use this to classify images.

The traditional neural network also performs admirably for this task, albeit taking far longer to train and at a lower accuracy.

In comparison to the relatively short times to run the KNNs, the neural networks take hours to train, but at the same time only have to be trained once. With this tradeoff comes far better accuracy and real-world usability of the predictions across both datasets. Our features-based KNN is by far the fastest, but it loses out by all reasonable metrics as far as precision was concerned. We conclude that the CNN is the best-suited method for the task of character recognition among those we tested.

V. References

[1] P. Frey, D. Slate, "Letter Recognition Using Holland-Style Adaptive Classifiers," Machine Learning, Springer, March 1991, Volume 6, Issue 2, pp. 161–182. https://doi.Org/10.1007/bf00114162

[2] T. Makkar, Y. Kumar, A. K. Dubey, Á. Rocha dan A. Goyal, "Analogizing time complexity of KNN and CNN in recognizing handwritten digits," in International Conference on Image Information Processing (ICIIP), Shimla, 2017. https://ieeexplore.ieee.org/document/8313707