CS 624: Final Project Report

McKayla Hagerty
Computer Science Department
Norfolk, USA
mhage001@odu.edu

1 Abstract

The project was designed in response to a Kaggle challenge on digit recognition. The goal of the challenge is to build a model to predict the classification of images of handwritten digits [1]. To make the project more interesting, I decided to create and compare three different types of models: K Nearest Neighbor (KNN), Random Forest Classification, and Convolutional Neural Network (CNN). What is the best implementation of each of the three algorithms and which model provides the highest accuracy on the test data? Kaggle Challenge: https://www.kaggle.com/c/digit-recognizer/overview

2 Keywords

Machine Learning, Computer Vision, Digit Recognition, Big Data, Convolutional Neural Networks

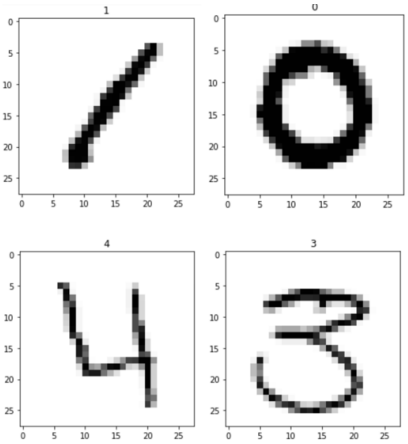
3 Problem Statement & Background

The goal was to compare appropriate classification algorithms to identify digits from a dataset of handwritten digits. Success is measured by the proportion of digits in the test dataset correctly classified. I decided on comparing KNN, Random Forest, and CNN algorithms. What is the best implementation of each of the three algorithms? KNN and Random Forest were quickly up and running after a few experiments to determine parameters. My CNN model was more extensive in its experimentation and implementation process. In each case, I ran experiments to make an educated decision on how to set up the model. Which model provides the highest accuracy on the test data? The metric I used to compare the final models was in accuracy of classification of the test set provided by Kaggle.

4 Approach

The training data provided by Kaggle includes 42,000 rows and 785 columns. The first column called label contains the digit intended to be written and each digit is followed by 784 cells mathematically representing the handwritten image through pixel values. As seen in Figure 1, each image is 28 pixels high by 28 pixels wide (28 pixels x 28 pixels = 784 columns). The larger the pixel value, the darker the pixel which indicates pixels that the handwritten digit goes through. The test data includes 28,000 rows representing 28,000 handwritten digits. The test data in set up the same way as the training data except without the label column. The pixel values range from 0 to 255. I normalized the data by converting them to values between 0 and 1 to simplify the analysis.

Figure 1



To run experiments on my models and get an idea of the expected Kaggle accuracy score of each variation, I split the training data into training and validation sets. Since this is a large dataset, I chose to only pull out 10% of the training data for the validation set. This allowed for a significant validation set of 4,200 digits and still left 37,800 digits to be used to train the models.

My initial research more than hinted at the known success of CNNs in computer vision problems. I knew I wanted to implement that algorithm, but I did not initially know what algorithms to compare it to. Once the data was ready, I used the lazyclassifier library to determine which machine learning methods were worth investigating father. Based on the results, I chose to focus on methods that I was familiar with and showed promise of success: KNN and Random Forest Classifier. By adjusting their parameters, my goal was to replicate, and potentially improve, the accuracy provided by lazyclassifier. I then compared my final model for each against the one less familiar to me, CNN. In pursuit of the best model, I ran experiments on several parameters for each type of model as described in the following sections.

4.1 K Nearest Neighbor (KNN)

The two parameters I adjusted to decide on my final KNN model were the number of neighbors and weights. For the

number of neighbors, I considered values from 1 to 10. The weight types I tested were ‘uniform’ and ‘distance.’

‘Uniform’ weight means data points in the full neighborhood are weighted equally in the determination of the classification of an unknown value. ‘Distance’ weights means closer neighbors will have a greater influence on how the unknown is classified. I only ran this experiment once rather than taking the average of 5 runs due to the significant amount of time required to predict using KNN. My results are shown in Figure 2. Based on the findings, I decided on 4 neighbors and a distance calculation for weights. My choice was also supported by the comparison of the weight options in their train and prediction times with the uniform option taking slightly longer for both (Figure 3).

Figure 2

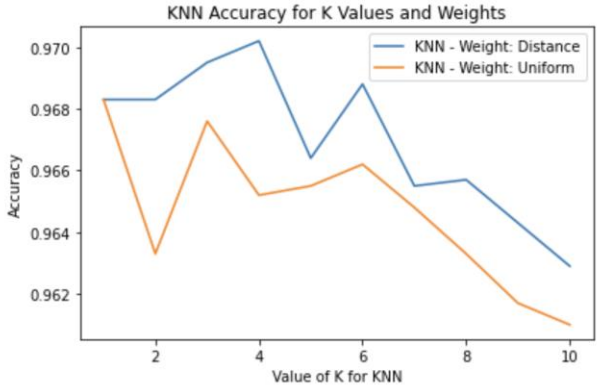
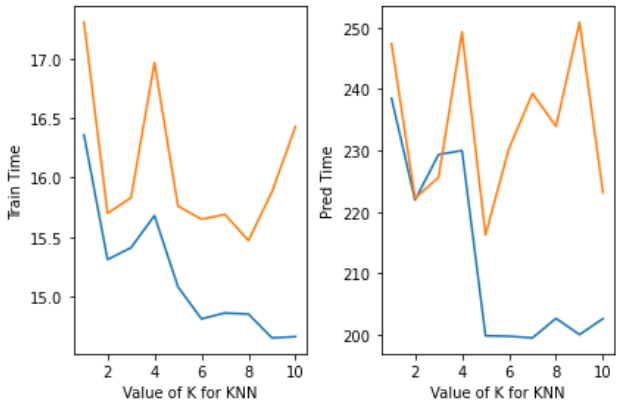


Figure 3

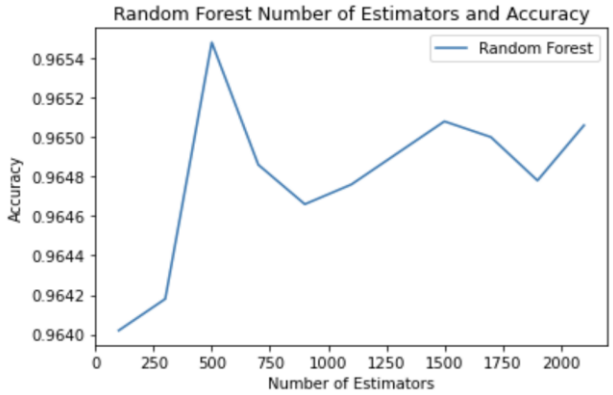


The final KNN model was fed the entire original training set before being used to predict the digits in the provided test data. The original training set includes both the train and validation sets. This full training data fitting was done for all final models throughout the project. The model earned a Kaggle accuracy score of 97.07%. It most frequently confused 9s and 4s as well as 1s and 7s. However, for such an easy model to set up, it preformed well. Since this model does not involve training, it is slow to predict, especially with large data, because the training data must be search each time a prediction is needed. This also means all the training data must be stored. Overall, it was a simple to understand, effective first model.

4.2 Random Forest

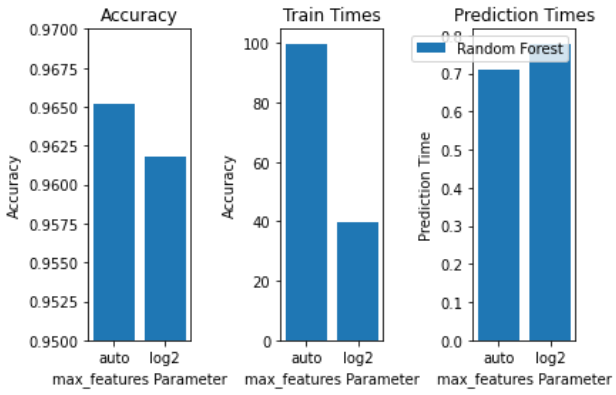
The two parameters I adjusted to decide on my final Random Forest model were the number of estimators and the maximum number of features. For these experiments, I ran each 5 times and took the average for more accurate results. The number of estimators (n_estimators) is the number of trees in the forest. In my experiments, I tested the following values: 100, 300, 500, 700, 900, 1100, and 1300. I decided on 500 trees because of the high accuracy and as the number of trees increased, the train and prediction times linearly increased.

Figure 4



For maximum number of features (max_features), or the number of features to consider when looking for the best split, I tested the ‘auto’ (max_features=sqrt(n_features)) and ‘log2’ (max_features=log2(n_features)) options. As seen in Figure 5, while the train times were longer with the default ‘auto’ option, the accuracy was higher, and the prediction times were lower. For my final model, I decided that some extra train time was worth the increase in accuracy.

Figure 5

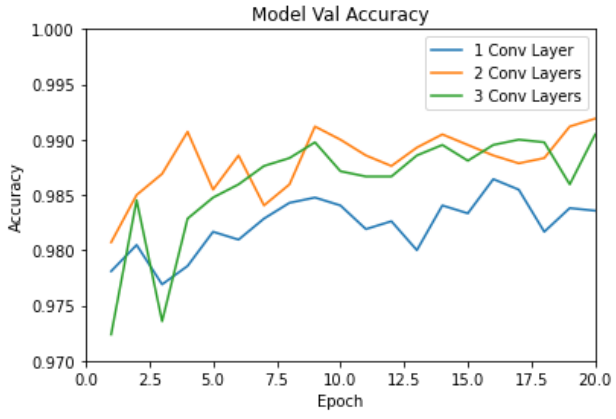


In comparison to the KNN model, the Random Forest model had a more even proportion of incorrectly and correctly classified digits for each digit. The model earned a Kaggle accuracy of 96.80% after being trained with the entire training set. This is a slight decrease from the KNN accuracy, but still an effective, easy to get running model. Random Forest models tend to work well on large datasets with high dimension. They also have reduced error in comparison to decision trees because they take input from all trees, in this case 500 trees, making them less prone to overfitting. Unfortunately, while Random Forest models are easy to set up, it's difficult to know or control what is happening during the formation of the forest.

4.3 Convolutional Neural Network (CNN)

Since this project marked my first exposure to neural networks, I spent two weeks learning the basics of neural networks and CNNs and researching follow-up questions before creating my first CNN model. The structure and parameters of my first model were very standard: 2 convolutional layers with decreasing filter size and kernel sizes of 3 followed by max pooling and flattening. This vaguely educated structure preformed better than my final KNN and Random Forest models with 98.14% accuracy on the test set. My next steps were to see if I could improve the model with more intentional structure and hyperparameter choices. My first decision was in the number of convolutional layers to include. I set up three simple models (model 1 with one layer, model 2 with two layers, and model 3 with three layers) and compared their accuracies (Figure 6). Knowing that adding additional layers results in increased computational cost, I decided on 2 convolutional layers. Three layers did not significantly increase the accuracy. I also noticed some validation loss increases in some of these initial experiments. This led me to believe that there may be a tendency of overfitting. I noted adding some dropout layers in my final model may increase the validation and final test accuracy.

Figure 6



The structure included 2 rounds of a convolutional layer, max pooling, and dropout followed by flattening and two dense layers. The convolutional layers are the heart of the model. They contain the weights that are adjusted with each epoch. Pooling replaces the output with a summary statistic, in this case the maximum value, of the nearby inputs in order to reduce the dimensionality. The dropout layer was a response to my suspicion of overfitting in my previous CNN models. It randomly sets a certain percent of input units to 0 at each step of training [2]. I then used the Keras Hyperband tuner with early stop monitoring the validation loss to determine the hyperparameters of the model. Using the parameters provided by the tuner, I received a Kaggle accuracy of 99.07%.

CNNs are excellent at automatically detecting the important features and capturing patterns which makes them a smart choice for image classification. Since CNN is deep learning algorithm, it improves as more training data passes through. As a final step, I decided to use data augmentation to increase the training data available to pass through the model. Data augmentation allows for a different transformation of each image in each epoch. Since the model is run for 9 epochs, 9 different versions of each original image in the training set were created and used [4]. These efforts increased my Kaggle accuracy by 0.3%, bringing it up to 99.37% (Figure 7).

4.3 Final Model Accuracy Comparison

The final model accuracy scores from my Kaggle submissions are summarized in Figure 7. These models were all trained (or in the case of KNN, stored) on the full original training set before being used to predict the digit classes of the test data.

Figure 7

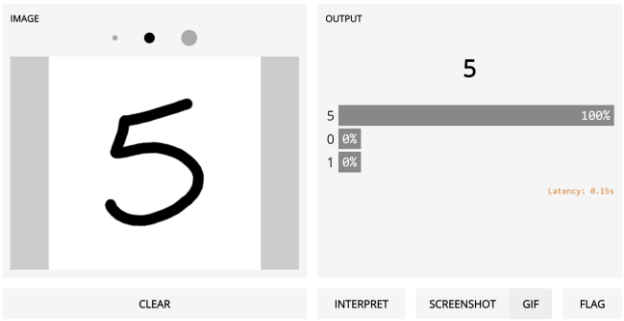
Model	Accuracy
Final KNN	97.07%
Final Random Forest	96.80%
CNN fit with original training data	99.07%
CNN with data augmentation	99.37%

5 User Interface

After finalizing my best model, I used gradio to create a machine learning demo user interface integrated directly into my Python notebook. This allows users to input their own handwritten digits and automatically have my final model (CNN tuned and trained with augmented data) predict the class [3]. The interface does not currently crop the images so they must be drawn close to the center and a similar size to what is shown in figure 8. This ensures

accuracy close to the test data accuracy by providing new data as similar to the training data as possible. The Kaggle test and train data was provided already cropped and resized.

Figure 8



6 Conclusion

My project problem statements were:

1. What is the best implementation of each of the three algorithms (KNN, Random Forest and CNN)?
2. Which model provides the highest accuracy on the test data?

Throughout the project, I discovered and implemented parameter adjustments to increase the accuracy of each model while keeping computational cost in mind. I also found that my original research was correct; the CNN models were most effective in this computer vision application.

The area with the largest increase of professional growth was in my comfort working with large amounts of data in python. From normalizing to reshaping and splitting, my comfort and confidence with python and debugging grew. I became more comfortable with KNN and Random Forest, algorithms with which I previously had minimal experience. I discovered the benefits of each and how to run experiments on them to determine the best parameters. Ultimately, I was introduced to CNNs and spent many hours learning how those models worked and why they worked better than my other models. I was able to further improve my original CNN model through hyperparameter tuning and data augmentation. The results were as expected from initial research. Even my simple initial CNN model outperformed my final KNN and Random Forest models, but by going through the process, I was able to understand the why and how. I am now more prepared to take on more advanced computer vision problems.

References

[1] Digit Recognizer Competition. (n.d.). Kaggle. Retrieved February 1, 2021, from https://www.kaggle.com/c/digit-recognizer/data

[2] Team, K. (n.d.). Keras: the Python deep learning API. Keras API. Retrieved March 1, 2021, from https://keras.io/

[3] *Gradio*. (2020, November 4). Gradio. https://www.gradio.app/

[4] Does ImageDataGenerator add more images to my dataset? (1967, May 01). Retrieved from https://stackoverflow.com/questions/51748514/does-imagedatagenerator-add-more-images-to-my-dataset