Classification on Modified MNIST Digit Dataset

Oliver Clark - 260674845,
Matthew Etchells - 260680753
Jonathan Pearce - 260672004
March 18 2019

***Abstract*—The MNIST dataset is the most well known and most commonly used introductory dataset for computer vision classification. The MNIST classification task is identifying what digit (0-9) is present in an image. In this project we work with a modified MNIST dataset where each image contains multiple digits, and where the task is to find and classify the largest digit in the image. In this report we outline our image processing technique to find the largest (in terms of pixel size) digit in each image. We discuss our design and validation of four different classifiers; random forests, k-nearest neighbors, a two layer fully connected neural network, and a convolutional neural network (LeNet-5). A basic bounding box technique was relatively successful at extracting the largest digit in each image. For the classification task, the neural network models were the most successful, with the LeNet-5 convolutional neural network achieving our highest validation accuracy of 94.35% and 94.266% accuracy on a subset of the test data.**

CONTENTS

I. INTRODUCTION

The Modified National Institute of Standards and Technology (MNIST) dataset was originally released in 1998 [11]. It was released as an updated version of the 1995 NIST dataset. The dataset is composed of 28x28 pixel grey-scale images each with a handwritten digit centered in the image. Its most common use being the development and training of handwritten digit recognition models. MNIST is among the datasets of choice when evaluating novel machine learning algorithms in the realm of object character recognition and computer vision.

In this project, we were provided a modified MNIST dataset with 40,000 and 10,000 images in the training and test sets respectively. Each image in our dataset had more than one digit, unlike the MNIST dataset. Our goal was to train a model to compute the largest (pixel-size) digit in an image. The modification complicates the problem from that of the original MNIST classification, as we now must consider multiple (possibly overlapping) digits present in images, for which the largest must be selected and classified. Our approach to this more complex problem was to first preprocess images using computer vision techniques to initially extract the best guess at the largest digit and then to train machine learning classifiers on the corpus of extracted digits from step 1.

To find the largest digit in each image we converted the grey scale images to binary images (pixel intensities of 0 and 255 only). We then proceeded to find the contours in the image and built bounding boxes around each contour. Next we found the largest bounding box in each image, with some exceptions due to our classifier input requirements and digits that were tangent to each other in certain images. We then centered the bounding box pixel values onto a new 28x28 pixel image which served as our processed image.

For the second step we experimented with four different types of classifiers that have been proven effective with digit recognition; random forests, k-nearest neighbors, a two layer fully connected neural network, and the LeNet-5 convolutional neural network. Using an 80/20 training-validation split on the training data we were able to evaluate and compare each model accurately. The random forests and the 2 layer neural network performed adequately, achieving 90.96% and 91.09% accuracy on the validation set. Similarly to previous research, the LeNet-5 convolutional neural network [11] was the strongest classifier achieving a validation accuracy of 94.35%. We selected the LeNet-5 model for submission to the Kaggle Competition, on 30% of the test data our model earned a 94.266% accuracy.

II. RELATED WORK

The MNIST data set was originally created and tested by LeCun et al. [11] in 1998. In their paper they test a number of different types of machine learning models on the dataset, providing an accurate comparison between techniques both in terms of model accuracy and compute power required in

training and testing. Previous work, especially that by Lecun et. al, shows the incremental improvement in accuracy on the MNIST dataset. Early approaches such as K-nearest neighbors and support vector machine found success, whereas recent work has been dominated by the success of convolutional neural networks or other deep learning methods [5]. However, the original MNIST dataset presents the problem of object classification. Our task involves digit localization by bounding box prediction as well as classification. Previous work has similarly looked at datasets of images that contain multiple digits as part of a mobile application for digit recognition [15]. This work by Yang similarly found best success using convolutional neural networks for the classification task, after processing the image of multiple digits into individual segments.

III. Dataset and Preprocessing

The data for this project was a modified MNIST dataset comprised of grey-scale images of size 64 x 64 pixels. Each image contained more than one digit, with the goal to find and classify which digit occupies the most space in the image. The notion of how much space a digit occupied was defined by a square bounding box fit around the digit. The dataset contained 40,000 and 10,000 images for the training and test sets respectively.

The original task of this project had two clear objectives, find the largest digit in the image and then proceed to classify that digit according to its numerical value. Based on other teams earning very high accuracy's (≈93-98%) early on in the competition and the limited time frame for this project we decided to pre-process the images to find the largest digit. Thus, reducing our model's objective to only having to classify a single digit's numerical value.

Each image in the dataset had a significant amount of pixel value noise in the background. This noise varied significantly between images both in pattern and pixel value. In order to remove the noise each grey-scale image was transformed into a binary image using OpenCV's [3] `threshold` function with a threshold value of 250. This transformation made the digits in the image black and the background white. Next we used OpenCV's `findContours` and `boundingRect` functions to construct bounding boxes around each digit, similarly to the image preprocessing work in [15]. The standard MNIST dataset contains 28 x 28 pixel images, therefore we decided that capturing the largest digits within a 28x28 pixel space would make our model development easier as we could more easily utilize prior research to aid our work. Next, we found the bounding box in each image with the largest length or width (since the space occupied by the digit was defined by a square bounding box) while still being less than or equal to 28 pixels in magnitude. We then centered this bounding box on a uniform white (0 pixel value) 28 x 28 pixel image. Thus producing a 28x28 binary image featuring the largest digit from the original grey-scale image. There were 493 images across the training and test sets with a bounding box dimension larger than 28 (≈1% of data), however many of these were from images with digits that were very close together and our methods above did not split them into separate bounding boxes. We opted to not manually annotate these cases, as it would take a significant amount of time. There were 27 images that did not contain any bounding box smaller than 28x28 pixels (≈0.05% of data), in these cases we took the bounding box that OpenCV found first and concatenated it to a size that fit within a 28x28 pixel image. The new training and test sets contained 40,000 and 10,000, 28x28 binary images respectively, with each image containing one single digit.

IV. Proposed Approach

Using the work of Lecun et al. [11] as our motivation, we selected three models from their paper to train and test on our now simplified image data, as well as one alternate model not mentioned in their work. The only data processing was to normalize the intensity values to [0,1], since our images were binary, 0 mapped to 0 and 255 mapped to 1. All model training was completed with 80% of the training data, and model validation used the remaining 20% of training data. Our digit classifiers included the following models:

A. Random Forests Model

A random forest classification model (RFC) was implemented with the Scikit-learn python library using the `RandomForestClassifier`[1] function. The random decision forest algorithm was first proposed in 1995 by Tin Kam Ho. Ho's work used the original NIST dataset to prove the validity of the random decision forest [7]. Random forests is an ensemble learning technique [7]. In testing we varied the 'number of estimators' parameter in the model, this parameter represents the number of weak learners present in the model. We experimented with values of 10, 100 and 1000. All other parameters for our random forests model were the default Scikit-learn parameters. Our accuracies using the random forest classifier are reported in Section V Table I.

B. K-nearest neighbors Model

K-nearest neighbors [2] is a popular instance based learning algorithm for classification tasks such as ours. The model works by finding the k 'closest' instances from the training data for each data point in the test set. The algorithm then finds the majority class among the k neighbors and that class is used as the prediction. The most popular formula for the distance between data instances is the Euclidean distance (L2), LeCun et al. utilize this distance measure in [11], however the L3 distance formula has been used with K-nearest neighbors and the MNIST dataset before. We choose K-nearest neighbors as one our first models because in [11] LeCun et al. use their K-nearest neighbors' model's performance as their baseline for the rest of the paper. We implemented our K-nearest neighbors model with the Scikit-learn python library [13] and their built in `KNeighborsClassifier` [2] function. We varied the k

[1]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[2]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

parameter in the model, where k represents the number of nearest neighbors that are chosen to help make the class prediction. We experimented with k values of 3,5 and 10. All other parameters for our K-nearest neighbors model were the default Scikit-learn parameters. Our accuracies using the K-nearest neighbors model are reported in Section V Table II.

C. Neural Network Models

Neural networks [6] are a type of machine learning model, whose design and structure is inspired by the biological neural networks found in the brains of living creatures, such as humans. A standard neural network contains a one dimensional input layer of nodes where data such as pixel intensity values (in our case) are each fed into their own input node, a number of connected hidden layers each with their own nodes that are connected to nodes from the previous layer and to nodes in the next layer and then finally an output layer with its own nodes. Each node in the network has the ability to receive input (from nodes in the previous layer), transform that input according to some activation function and set of weights that belong to that node and output the result of that computation (to nodes in the next layer). The outputs from the output layer nodes can be interpreted as a model prediction.

1) 2-Layer Fully Connected Neural Network Architecture: LeCun et al. [11] were able to improve upon the accuracy of their k-nearest neighbors baseline using a two layer neural network. Following this path appeared to be a feasible entry point into network architecture for this project. Using Keras [4] with a Tensorflow [1] backend, we designed a two layer fully connected neural network. Fully connected means that each node in layer k is connected to each node in layer $k+1$. Our input layer contained 784 nodes (28^2), hidden layer 1 contained 512 nodes, hidden layer 2 contained 128 nodes and the output layer contained 10 nodes. The activation function for the hidden layers was the ReLu [12] function and on the output layer we used a softmax activation function. Finally for each hidden layer, to help prevent network overfitting we used dropout regularization [14] with a probability, $p = 0.5$ of 'dropping out' a unit for both layers. For training we used the adam optimizer [9], all other network training parameters were the default Keras values.

2) LeNet-5 Convolutional Neural Network Architecture: For our final model we implemented the LeNet-5 convolutional neural network from [11]. Convolutional neural networks [10] are a type of neural network designed to work well with image data. As opposed to the regular neural networks discussed in the section above, convolutional neural network layers are capable of accepting three dimensional input (tensors), and outputting transformed three dimensional data. Using local receptive fields convolutional neural networks are able to find distinct image features and pooling/subsampling helps reduce the network dimensionality, layer by layer. Convolutional neural networks are able to effectively handle the high dimensions associated with image data, exploit the 2D topology of pixels in images and can be designed to be invariant to certain expected variations between images such as translations and illumination differences. The focus of [11] was to introduce the LeNet-5 convolutional neural network architecture. In the paper LeNet-5 recieves the lowest test set error rate of any model using the standard non-distorted MNIST data. In [11], the images are 32x32 pixels, therefore our first step in our implementation of LeNet-5 was to zero pad the image to create 32x32 pixel image. The LeNet-5 architecture contains seven network layers. Layer 1, is a convolutional layer with 6 channels, a 5x5 kernel, and a tanh activation function. This layer accepts a 32x32x1 tensor and produces a 28x28x6 tensor. Layer 2 is for subsampling, it is uses average pooling with a 2x2 filter, the output is a 14x14x6 tensor. Layer 3 is the second convolutional layer with 16 channels, a 5x5 kernel, and a tanh activation function, the output from this layer is a 10x10x16 tensor. Layer 4 is another subsampling layer, it is also uses average pooling with a 2x2 filter, the output is a 5x5x16 tensor. This tensor is flattened into a 400x1 vector and fed into the last 3 layers of the network. Layers 5,6 and 7 are fully connected standard neural network layers with 120, 84 and 10 nodes respectively. The activation functions for layers 5 and 6 were tanh and finally layer 7's activation function was softmax. For training we used the adam optimizer [9], all other network training parameters were the default Keras values.

V. Study Results

Our initial experiments evaluated the predictive accuracy of non-deep learning approaches including random forest and K-nearest neighbors classification models. We aimed to establish a base line model and validation accuracy for comparison with deep learning approaches.

We trained random forest classifiers using the sklearn library's [13] implementation, varying the `num_estimators` parameter which describes the number of weak learners used in the ensemble. Results are shown in Table I. We found highest validation accuracy when using 1000 weak learners.

Estimators	**Accuracy**	**Time (train) (seconds)**	**Time (validate) (seconds)**
10	0.86775	1.305	0.065
100	0.903625	12.104	0.411
1000	0.909625	126.893	6.098

TABLE I: Random forest validation accuracy and run time, varying number of estimators

Similarly, we trained K nearest neighbors classifiers, also using the sklearn implementation, varying the `num_neighbors` parameter. Results are shown in Table II. We achieved highest accuracies using 3 and 5 neighbors, though results were very similar for 10 neighbors as well. Given that the k nearest neighbor classifier is a form of instance based learning, training (fitting) of the model is a less laborious task than making predictions, which requires training sample lookup. This is reflected in the run times as

the validation (prediction) time is significantly greater than the time necessary to train the model.

Neighbors	Accuracy	Time (train) (seconds)	Time (validate) (seconds)
3	0.88925	7.247	354.493
5	0.88925	9.651	321.026
10	0.887	7.678	335.247

TABLE II: K-nearest neighbors validation accuracy and run time, varying number of neighbors

Among our initial experiments the best accuracy found on the held out validation set was 90.96% when using a random forest with 1000 estimators. This model and accuracy served as a baseline comparison for subsequent deep learning experiments.

The following Table III shows the performance of both the 2-layer and LeNet-5 neural networks. Figure 1 shows accuracy of the LeNet-5 model for both the training and validation sets for a varying number of epochs. After each epoch we would evaluate the model on the validation set. When we encountered the situation in which epoch $k+1$ resulted in lower accuracy than epoch k, we considered that the model from epoch k was the best possible before overfitting. We found the highest accuracy after 13 epochs of training. After the 13 epoch mark, training accuracy continues to improve where validation accuracy gets worse, indicating that the model is overfitting.

Model	Training Epochs	Train Acc.	Valid. Acc.	Test Acc.
2-layer NN	16	0.8884	0.9109	-
LeNet-5	13	0.9463	0.9435	0.94266

TABLE III: Neural Network number of epochs in training and training, validation and testing accuracy

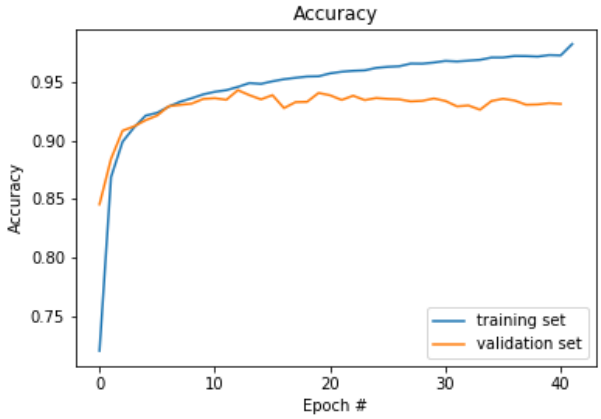

Fig. 1: LeNet-5 accuracy of training and validation set

Using the LeNet-5 architecture LeCun et al. were able to achieve 99.05% accuracy with the unmodified MNIST dataset. Ideally, we would be able to recreate this accuracy on the modified MNIST dataset through image preprocessing. In analyzing the images incorrectly classified by our model, we determined there are three cases yielding an incorrect prediction; (1) the neural network predicts incorrectly (see Figure 2 below) (2) the image preprocessing misidentifies the largest digit (due to noise or digit overlap), prohibiting the neural network from making an accurate prediction (see Figure 3 below). We were able to achieve 94.26% accuracy, indicating that our image preprocessing introduces at most 4.8% reduction in accuracy.

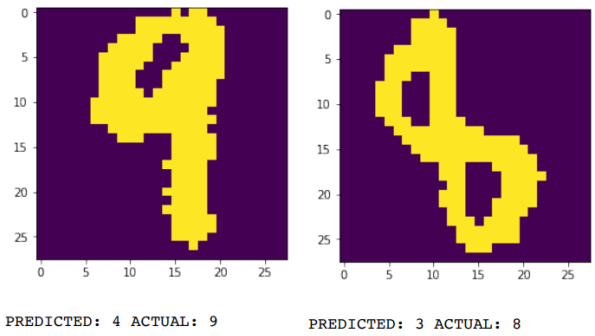

Fig. 2: Incorrectly Classified Images

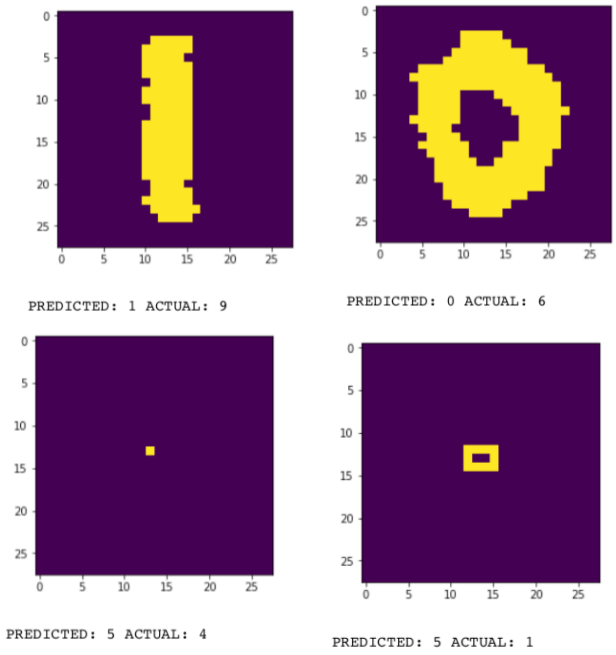

Fig. 3: Incorrectly Selected Bounding Boxes

VI. Discussion & Conclusion

We found that the best validation and test accuracy among our selected models was given by the convolutional neural network with LeNet-5 architecture, consistent with best approaches in previous work [11].

In our project, we considered a 2 step approach. The first

was to use the openCV [3] computer vision library to draw square bounding boxes around objects in the image and to crop out the largest of these boxes. The second was to train machine learning classifiers on these cropped images.

The most significant limitations in our approach may be from poor labeling of samples extracted from the first step, in which we used image contouring. The method of contouring involves clustering of similar pixels, although many samples in the corpus contained significant noise or even overlapping of digits that may have been clustered into bounding boxes that were extracted as the largest. The limitation is that these inaccuracies in finding the largest bounding box were then transferred into incorrectly labeled training samples. The machine learning models were thus training on imperfect data. We expect that the effects of mislabeling did not have major impact, as the majority of samples were correctly labeled. However, improving the step of digit recognition would likely bring us closer to accuracies observed on the original MNIST dataset. Surely our approach was unable to properly label noisy or difficult samples correctly. We may have considered a more sophisticated method of labeling. For example, the method of contouring is generalizable to all objects in images. We may have considered a labeling method that was specific to MNIST digits. A model may have been initially trained on the original MNIST dataset, and used for detection on our modified dataset. We may expect better results from a model that is specifically trained on MNIST digits for localization and bounding box prediction, than one that is generalizable to all objects. We may have considered other additions to our image processing, for example, we may have tried to normalize the location of digits in our training example. We may have also used methods described by [8] to orthoganally transform the image such that the variance is maximized.

Future work could use a more complex architecture; LeCun et al. was published in 1998 and classification techniques and architectures have improved in the past 20+ years. Ciresan et al. [5] were able to achieve 99.73% accuracy using a committee of classifiers on data that was preprocessed or normalized in a variety of ways. Implementing this more advanced committee of classifiers could have been more invariant to image preprocessing yielding higher accuracies. We may also consider and end-to-end approach to this problem of digit detection and classification, in which a model could be fed in the modified MNIST data and be trained on these images without the need for image preprocessing. Such end-to-end approaches have gained attention in recent years.

Regardless of potential future improvements, our approach still proved successful on the vast majority of test data, achieving a score of 94.266% on the available Kaggle test set.

We consider also the trade off between the amount of time required to train models, and the validation accuracies that they are able to achieve. Although we find the highest validation accuracies with neural network approaches, their training times were significantly greater than our best results with random forest classification. Each epoch in training the neural network models required approximately 1 minute, and we found the best results when training for 10-13 epochs. The random forest on the other hand was able to reach 90% accuracy after only 12 seconds of training. Assuming that these findings are generalizable, we may consider the use of the random forest classifier in situations where the data set is much larger, compute power is scarce and 90% accuracy is considered acceptable.

In conclusion, our findings are consistent with previous work in that deep convolutional neural networks provide the best accuracy for the MNIST digit classification task. We also note that our approach suffered from limitations in image preprocessing, which may explain lower accuracies that have been noted in previous work. In the future we consider more advanced image processing methods, specific to MNIST digits in images, or an end-to-end approach that combines the tasks of finding the largest digit in the image of many and classifying within a single model or ensemble.

VII. Statement of Contributions

All three members worked on the image processing code. Oliver wrote the absract, introduction and related work sections. Matt wrote the code for the random forests and k-nearest neighbours models and wrote the discussion and conclusion. Both Oliver and Matt worked on the results section. Jonathan wrote the code for the neural network python notebook and wrote the dataset and proposed approach sections.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

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

[4] F. Chollet et al. Keras. https://keras.io, 2015.

[5] D. Ciresan, U. Meier, L. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. pages 1135 – 1139, 10 2011.

[6] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[7] T. K. Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–, Washington, DC, USA, 1995. IEEE Computer Society.

[8] I. Jolliffe. *Principal Component Analysis (Springer Series in Statistics)*. Springer, 2002.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[10] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[12] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[13] 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.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[15] X. J. Yang. Mdig : Multi-digit recognition using convolutional nerual network on mobile. 2015.