COMP 551 - Applied Machine Learning - Project III: Image Analysis on the Modified MNIST Dataset

Afuad Hossain
afuad.hossain "at" mail.mcgill.ca
260621073

Jonathan Adalin
jonathan.adalin. "at" mail.mcgill.ca
260636904

Parker King-Fournier
parker.king-fournier "at" mail.mgill.ca
260556983

I. Preface

To access code related to the mentioned algorithm implementations in this paper, simply visit the Github Repository: https://github.com/JonathanAdalin/comp551-project3

The team name at https://www.kaggle.com/c/comp551-modified-mnist/leaderboard is Parker King-Fournier.

II. Introduction

Recent advances in machine learning algorithms have provided for substantial gains in accuracy for crucial tasks including image classification. Through the use of deep learning methods such as convolutional neural nets, accuracy numbers on otherwise complicated analytical tasks have gone so far as to even outperform humans.

This paper highlights the build process of an image analyzer which automatically computes mathematical functions from images. The dataset, a modified class of the MNIST dataset, consists of images that contain two hand-written digits and an operation denoted by either the letter a (or A) for addition or the letter m (or M) for multiplication. In total, there exists 40 distinct classes of answers to all combinations of digits and operation. Substantial image preprocessing was required to have images readily classified without much ambiguity. Classifier types utilized include a baseline logistical regression, feed forward neural net trained via backpropagation, and a convolutional neural net, the latter most successfully performing the task at an accuracy of 79.2%, a remarkable improvement from non-neural based ML algorithms.

III. Related Work

A plethora of research exists on building effective classifiers for the MNIST dataset. Accuracy rates for the classification of the hand-written digits database have been remarkable, with some groups having error rates as low as 0.5% when using deep neural net architecture (O'Shea, 2015). Furthermore, a relatively effective use of convolutional neural nets on the MNIST database was demonstrated with a classification accuracy of 92%(Blum, 2017). While this was lower than the accuracy rate of a fully connected neural net Blum utilized in the same research, the differences in feature diversity for the modified MNIST data set led us to believe, that with the proper preprocessing and feature extraction of the modified dataset, substantial gains are to be had from utilizing convolutional neural nets.

IV. Problem Representation

A. Dataset

The data used in this project is a modified version of the commonly know MNIST data set, and is significantly less user friendly than its predecessor. While both data sets contain grey-scale images, the images from Modified MNIST set show significantly more background patterning than the MNIST set. Additionally, the images in the MNIST dataset are pre-normalized and of constant size of 28x28 pixels. The Modified MNIST dataset, however, is un-normalized and each image is of the size 64x64: more than 4 times the size of the MNIST images

The images of the Modified MNIST dataset provide extra challenges due to the complicated background patterns and large image size in comparison to the MNIST set. Background patterns in the images require more rigorous approach to preprocessing the Modified MNIST data set so that it is usable. Ideally, all background patterns could be removed. The large size of the images may increase resolution, however the price to be paid will be in terms of computation speed. Preprocessing techniques will be discussed in the following section.

B. Data preprocessing

Preprocessing of the training and test data proved to be a crucial component of the success of the various classifiers. The goal preprocessing data was to reduce noise and increase clarity of the data, as well as to standardize the values and included only two steps: a normalization step and a binarization step based on a threshold value. The results of the preprocessing can be seen on sample images in Figure 1.

The normalization of the data was accomplished by sending the range of pixel values for image *I* from $[Min_I, Max_I]$ to [*0*, *1*] using the formula

$$I'_p = \frac{I_p - Min_I}{Max_I - Min_I}, \tag{1}$$

where I'_p is the normalized value of pixel p in image I and Min_I and Max_I are the minimum and maximum pixel values of image *I*, respectively.

After doing this, the first 200 images were visualized to note certain characteristics of the data could be that could be exploited to increase data quality. In doing so, two key features were recognized. It was observed that the images consisted

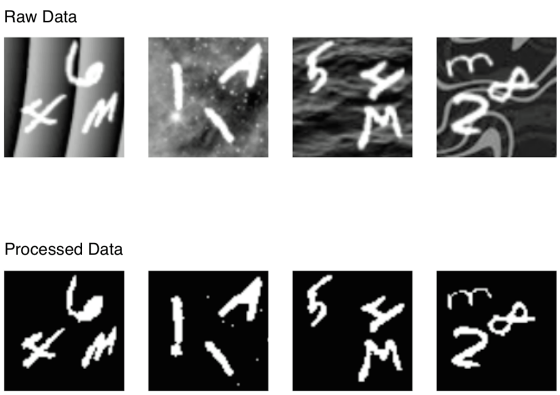

Fig. 1. A visualization of raw and processed image data.

each of two digits, one character, and various background patterns. Furthermore, the digits and characters of each image were largely white while the background patters were largely non-white. Thus, to eliminate background patterns a simple threshold binarization of the form

$$I''_p = \begin{cases} 0, & \text{if } I'_p \leq t \\ 1, & \text{otherwise} \end{cases}, \tag{2}$$

where I''_p is the result of normalizing and binarizing image *I*, and *t* is the threshold value.

In preliminary testing, a threshold value of *t = 0.97* was used as it appeared to create the clearest images. Later testing showed that, although a lower threshold value would leave more noise in the data, algorithms performed better when a lower threshold value was used. This can be understood by realizing a high threshold value may 'erase' characters that are not sufficiently dark. A lower threshold therefor allows more noise, but preserves the characters in the image more accurately. A value of *t = 0.7* was used in final testing.

Looking at Figure 1 shows some imperfections in the preprocessing techniques used. Various image processing techniques were explored to improve upon this. The second processed image shows small dots in the image left from the original background pattern. Morphological opening is a method which has the capability of removing small objects such as dots from an image (Serra, 1982). Although promising, the addition of morphological opening did not improve the quality of the data as the operation 'opened' characters in the image, often making them indistinguishable.

The fourth processed image in Figure 1 shows an 'm' with thin character strokes. This observation inspired the use of morphological dilation, an operation which has the capability of thickening (in this case) white character strokes. Again, it was found that the addition of this technique was unhelpful. While dilation increased definition in thin-stroke characters, thick-stroke characters became too dilated, resulting in the loss of distinguishing features such as the counters[1] or serifs[2].

After exploring these methods it was decided that the most effective preprocessing method was a simple normalization followed by a binarization with a threshold of *t = 0.7*.

Originally the processed data was stored as comma-seperated values (csv). The csv format required that Python construct an object from the csv file each time a classifier was trained. To speed up the reading in of data to classifiers the processed data was 'pickled'. Pickling is a Python specific method that converts a python object to character stream containing all information necessary to reconstruct the object in another python script. Becuse Python did not have to construct an object repetitively, saving data as pickles (pkl) allowed for faster importing of data when running classifiers.

C. Feature Design

Feature selection for the Convolutional Neural Networks (CNNs) proved to be straightforward. The features fed into the CNN were simply the 64^2 pixel values for each *I''*, as defined above. Thus, much of the feature design was handled through preprocessing the data. This exhibits an appealing aspect of deep learning which, in comparison to "shallow" learning methods, require very little feature design.

V. Algorithm Implementation

A. Logistical Regression

Utilizing readily available linear modeling tools in the sklearn toolkit, a logistic regression algorithm was implemented to classify the modified MNIST dataset. Images were input as arrays of 4096 values corresponding to the greyscale levels of the image. Cross-validation methods and other testing methods of sorts were intentionally omitted from the implementation process due to priorities principally being directed at deep learning architectures designs. Regardless of the lack of a validation process, default logistical regression settings, as outlined by the sklearn toolkit provided an accuracy rate of 12%, not surprising considering the tremendous amount of variables being processed by such an algorithm.

B. FeedForward Neural Net

The implementation of a Feedforward Neural Network was possible thanks to the COMP 551 course notes and the helpful guide "Implementing a Neural Network from Scratch in Python" by WildML.

The inputs to our neural network was the preprocessed array of bits stored in Pickle format that composed each image. The output was the index of the 40 possible outputs that we stored in an outcomes vector. We opted to use the tanh function as the activation function, which was recommended in the aforementioned article because of the ability to reuse calulated values to solve derivatives for gradient descent. Similar to what was taught in class, gradient descent was chosen to minimize the error at the output. Because of the significant time it takes

[1]https://en.wikipedia.org/wiki/Counter_(typography)
[2]https://en.wikipedia.org/wiki/Serif

to make the neural network learn, it was difficult to find the optimal architecture in terms of number of hidden layers and termination time. For this reason, we settled for 3 hidden layers and 1000 iterations of batch gradient descent with a fixed learning rate.

C. Convolutional Neural Net

The use of a Convolutional Neural Net was inspired by independent research in image recognition techniques. By far the most used algorithm in character recognition was a CNN, although the architectures varied with application. The architecture used can be seen in Figure 3.

The specific architecture used was chosen in regards to the task at hand. The images to be classified contain three characters, c_1, c_2, c_3, corresponding to a single output, o. While conventional Latin based languages use a character system in which only one character, c, corresponds to an output, o, non Latin based languages, such as Chinese, use a character system in which a character, C, often a collection of smaller characters, c_1, ..., c_n, corresponds to a single output, o. The similarity between Chinese characters and the images used in this project led to the adoption of CNN architectures similar to those that had been used to classify Chinese text.

The final CNN architecture chosen was inspired loosely by architectures described in literature relating to classification of Chinese text (Zhang) (Zhong & Jin, 2015). Architectures designed contained often upwards of 3 convolution and pooling layers, commonly two fully connected layers, and an output layer.

The CNN architecture used in this project was similar, containing two convolution layers, each followed by a max pooling, which fed into two fully connected layers, finally connecting to a softmax output layer. The first convolution layer used 40 filters, and the second used 64. After each pooling, the batch was normalized and passed to the next layer of the network. All weights were randomly initialized according to a truncated normal distribution with a standard deviation of 0.05, and all biases were initiated randomly according to a uniform distribution with range [0,5]. The two fully connected layers used a dropout rate of 0.5 to prevent overfitting. The CNN was trained using the tensorflow.train.AdamOptimizer()[3] method with a learning rate of 0.001 for 11,000 steps.

Originally an output layer of size 40 was implemented to match the 40 distinct classes. Later it was found that an output layer of 82 produced more accurate results.

VI. Results

A. Testing Methods

Cross validation of the CNN was accomplished by built in methods in Tensorflow. The model was trained on 75% of the training data, and validated on the remaining 25%. After a relatively acceptable score was gained, the CNN was trained on the entire dataset.

Classfier	% accuracy
Logistical Regression	12.0
Feedforward Neural Net	11.0
Convolutional Neural Net	79.2

Table 4. Results of Each Implemented Classifier, in order of accuracy

B. Results

VII. Discussion

A. Feedforward Neural Net

The implemented feedforward neural network performed at best with 11% success rate as we varied through different architectures by hand. It was known beforehand that the feedforward neural network would not be the best learning algorithm for image recognition, but there were variations to our methods that would have improved our results.

For example, instead of trying to find the best architecture in an ad hoc manner, this could have been determined through cross validation as suggested in the assignment instructions. Additionally, it was recommended by the WildML article to use a decaying learning rate over time to improve learning results.

Ultimately, these changes would improve our results only ever so slightly. These improvements do not come close to the extent of how changing to CNN can improve our image recognition capabilities. The latter is described in the next section.

B. Convolutional Neural Net

As expected, the CNN outperformed the other algorithms. Convolutional Neural Networks are widely used in image recognition, and offer many improvements over alternate classification techniques, notably Neural Networks. In traditional Neural Networks, a large number of neurons would be required to accommodate the large number of input values (pixels). CNNs reduce the number of free parameters in a given architecture and thus permit deeper networks than traditional Neural Networks.

A second advantage of CNNs over older techniques is the ability to learn filters. Traditional algorithms required that filters be designed by hand, leaving the quality of such algorithms to be dependent on the competency of those designing the filter. As CNNs do not rely on prior knowledge or human effort, they offer a huge advantage in classification, specifically of images.These advantages undoubtedly contributed the the success of the CNN over the Neural Network and Logistic Regression.

Through trial and error, a general trend of more layers resulting in an increase in prediction accuracy was observed. Improvement on the CNN architecture used would logically include increasing the depth of the network. Doing so would require access to greater computing power, as one of the limiting factors in progress was the amount of time that models required to train sufficiently.

[3]https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer

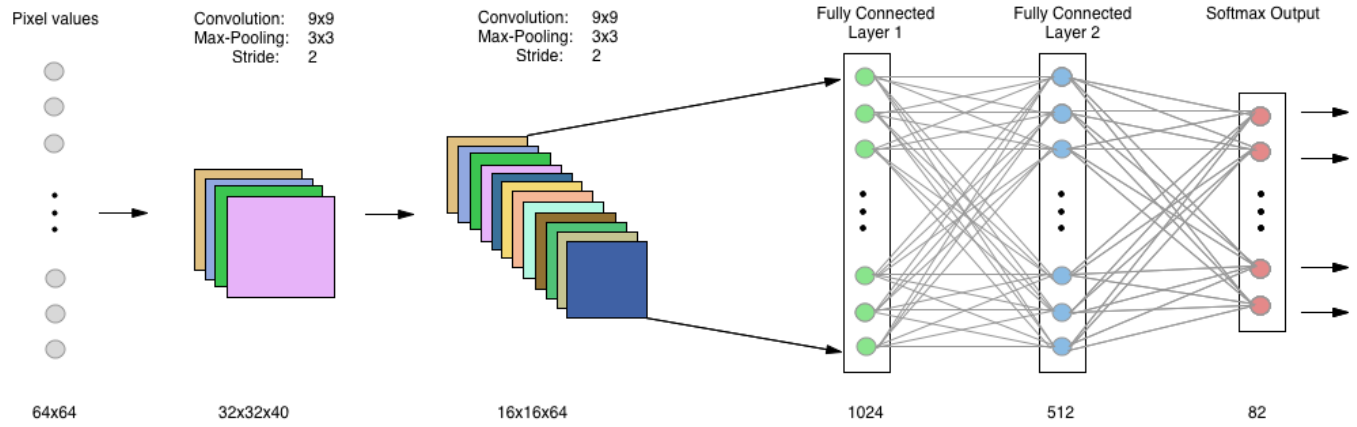

Fig. 2. Architectures of the convolutional neural network used. Labels above the diagram describe the layers. Labels below are the size of a layer's output.

Increasing the depth increased accuracy, but not to the extent that increasing the size of the original convolution filters did. A more effective CNN would ideally be deeper and utilize larger filters which shrink at each successive convolution layer.

The CNN could be further improved by implementing more effective preprocessing techniques. While it is true that CNNs require less preprocessing than other similar techniques, the value of clean, uniform data is paramount in the performance of any data-driven learning algorithm. A possible solution could be edge-smoothing of the characters, achieved with a Gaussian Blur.

VIII. Conclusion

The results of this research reveal interesting paradigms within the field of machine learning. Of particular interest is the low performance of the Neural Network, which is seen by many as the end-all be-all of machine learning approaches. This report shows that different learning and classification techniques will succeed at specific tasks and exhibits the importance of a knowledge of the pros and cons of each method. The use of the Modified MNIST dataset reiterated the importance of knowledge of preprocessing techniques specific to the dataset at hand. The hands on experience accompanying such a project is priceless, allowing one to confront and work through issues that arise in the real world while gaining an understanding of how machine learning is used in practice.

IX. Statement of Contributions

Throughout the project Parker King-Fournier was responsible for researching and implementing data preprocessing techniques. This included 'cleaning' the data as well as storing it in a format which was easily and quickly accessible to other programs and scripts. Parker also implemented the Convolutional Neural Network using the Tensorflow library in Python. Afuad Hossain implemented the logistical regression algorithm while providing necessary research into the implementation of the Convolutional Neural Net. Jonathan implemented the feed forward neural network algorithm.

All members contributed to the final report by writing the sections relevant to their work and reviewing the work of their teammates.

We hereby state that all the work presented in this report is that of the authors.

References

[1] C. W. Blum. "On the Effectiveness of Neural Networks Classifying the MNIST Dataset". Retrieved from the University of Minnesota Digital Conservancy, http://hdl.handle.net/11299/184865, 2017.

[2] K. T. O'Shea, "Massively Deep Artificial Neural Networks for Handwritten Digit Recognition", arXiv preprint arXiv:1507.05053, 2015.

[3] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[4] Y. Zhang,"Deep Convolutional Network for Handwritten Chinese Character Recognition". Stanford University, .

[5] Z. Zhong and Z. X. L. Jin, High performance offline handwritten chinese character recognition using GoogLeNet and directional feature maps, in Proc. Int. Conf. Doc. Anal. Recog., 2015. [Online]. Available: http://arxiv.org/abs/1505.04925

[6] J. P. Serra."Image analysis and mathematical morphology", London New York: Academic Press, 1982.

[7] WildML. (2017). Implementing a Neural Network from Scratch in Python An Introduction. Available at: http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/.