Training and Applying a Classifier with MNIST

Taylor Thurlow

Dr. Hao Ji, CS 4990 Fall 2018

CAL POLY POMONA

2018

ABSTRACT

The MNIST dataset is a set of handwritten numerical digits. This project leverages this dataset to train a convolutional neural network to identify new handwritten digits. The resulting model achieved an accuracy of around 99%, though there are some specific weak points of the model. Difficulty in utilizing this model is in the amount of preprocessing required for input data.

INTRODUCTION

The objective of this project was to built a multi-class classification model using the MNIST dataset. After getting the model to a reasonably high accuracy level, the next objective was to create my own testing data from my own handwriting, and determine what level of input preprocessing is required to achieve high accuracy results.

It should go without saying, but because the model is trained on a heavily preprocessed dataset, utilizing an MNIST-trained model will likely require an equal level of preprocessing.

Additionally, I was also interested in determining which numbers the MNIST-trained model would have issues with. With some direct observation, perhaps some conclusions can be drawn about the shortcomings of my model.

DATA

The MNIST dataset is a database of handwritten numerical digits available for free. It contains 70,000 labeled samples - 60,000 training and 10,000 testing samples. Each image has been heavily preprocessed relative to a simple photo of a handwritten digit on paper:

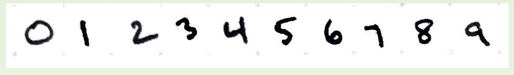
- Masking, creating a pure white mask on a pure black background
- Size normalization (introduces anti-aliasing, no longer all white/black)
- Centered by center-of-mass in a fixed 28x28 pixel frame

METHOD

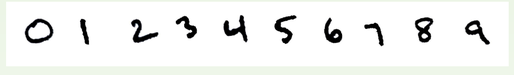
My model is made of 3 convolutional layers - the first two of size 32 and the third of size 64. Kernel size is 3x3. The model ends in a 256 dense layer to represent the final features, and after some dropout in order to avoid overfitting, ends in our final classification layer of size 10 (the number of classes). Softmax activation will give us one-hot encoded arrays which represent our classification result. 20% of the training dataset is used as validation - a total of 12,000 of 60,000 training images. A batch size of 16 was used, with a limit of 200 epochs (though with early stopping callback, it never got close to this limit). The Adam optimizer with default settings, as well as a reduce_learning_rate_on_plateau callback yielded the best results. Patience on all callbacks was set very high - The learning rate was reduced to below 1% of its starting value before training finally gave up. Code used is available on my GitHub.

To test the model with my own handwriting, I wrote each digit in black pen and took a photo with my phone. To preprocess the data, here are the steps I took:

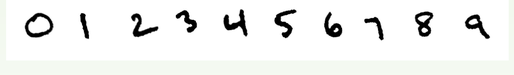
Photo taken with 'Adobe Scan' on Android



Levels adjustment to make whites whiter and blacks blacker, avoid as much grayscale as possible



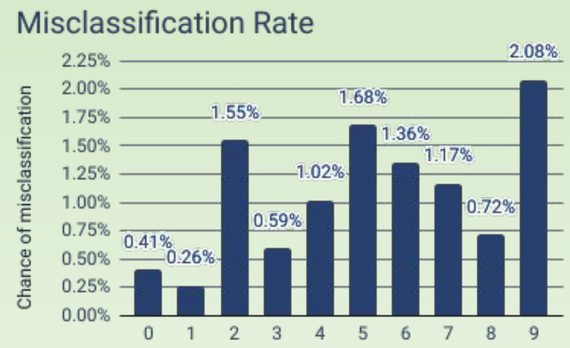
Add sharpness, re-convert to grayscale because sharpening introduces color



Invert color, slice into square images

RESULTS

Results of the training of the classification model were good, coming in at an accuracy of 98.93% Over the entire set of 10,000 testing images. Some classes were more troublesome than others - the model has a particularly hard time with 9s, and to a lesser extent, with 5s and 2s. There is a really large variance in writing style for the digit 9, many of the misclassifications being entirely understandable, even for a human. 2s were particularly interesting - the vast majority of misclassifications for the digit 2 were the digit 7. Visually these characters are quite similar, and the model has trouble identifying the flat bottom piece that defines the digit 2.



Providing the model with the images I generated from my own handwriting, it seems to do pretty well. It classified every digit correctly, except for 7 - it thought 7 was a 2. Clearly this 2/7 distinction is a shortcoming of my model.

Undoubtedly, testing the model with more of my own handwriting, especially handwriting done before I was aware that they would be used this way, would be interesting. Unfortunately the amount of preprocessing necessary is prohibitively time expensive. In order to preprocess any meaningful amount of data, some serious thought would need to be given to automating the process of identifying digits, applying the necessary visual filters, and selecting a bounding box.

CONCLUSION

My results reflect the thought and care put into constructing the MNIST dataset. While my model is quite accurate at almost 99%, this is likely insufficient for any real usage - use cases for handwritten digit identification are likely to be highly sensitive to mistakes. Areas of improvement may be in more tweaks to data augmentation, or perhaps smaller-sized convolutional layers. The real takeaway from this experiment, though, is definitely in the complexity of data preprocessing. MNIST's usefulness is limited by its users' ability to match input data to the kind of data MNIST was trained on.

REFERENCES

- Shrikar Archak, Deep Learning with Keras and Python for Multiclass Classification (2018) - https://shrikar.com/deep-learning-with-keras-and-python-for-multiclass-classification/
- Yash Katariya, Applying Convolutional Neural Network on the MNIST Dataset (2017) - https://yashk2810.github.io/Applying-Convolutional-Neural-Network-on-the-MNIST-dataset/
- Orhan Gazi Yalçın, Image Classification in 10 Minutes with MNIST Dataset (2018) - https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d