Classifying Modified handwritten Digits with Background Noise

551-001 - FreeSmoke

Nicolas Fortier
McGill University
Montreal, Quebec, Canada
260681624
nicolas.fortier@mail.mcgill.ca

Peter Park
McGill University
Montreal, Quebec, Canada
260571481
baekyun.park@mail.mcgill.ca

Yassine M'hedhbi
McGill University
Montreal, Quebec, Canada
260673563
yassine.mhedhbi@mail.mcgill.ca

***Index Terms*—Linear SVC, Convoluted Neural Networks, Feed-forward Neural Networks, image processing**

I. Introduction

This report documents our attempts at creating a Machine Learning model capable of recognizing the largest number through a noisy background and from a group of 2 to 4 candidates.

The entirety of our data was first fed through a preprocessing pipeline which removed all noise from the images, before determining which number was the biggest. Once that was done, a new image was created containing only the relevant number, as it appeared in the unprocessed image.

This data was used to train a Linear Support Vector Machine, a regular Feed-forward Neural Network and a Convolutional Neural Network.

Our kaggle competition result (96.985% accuracy) was obtained using our CNN. Our Feed Forward Neural Network had generally good performance (around 92% accuracy), while our Linear SVC model performed poorly (around 79% accuracy).

II. Feature Design

The raw input images were made very noisy on purpose. This, in and of itself, was very problematic to begin with.

Therefore, the first step in our path to a usable dataset was to remove all the background noise from our images. This was done very simply. Looking at the picture, it's clear that the pixels making up the numbers are much more activated than all other pixels. The activation values in our arrays ranged from 0 (pure black pixel in Figure 1) to 255 (pure white pixel in Figure 1). Therefore, all pixels with activation values under 200 were turned off. The result is shown below.

Now that our image is free of background noise, one key step remains: finding the biggest digit by area in the interim result. To do this, we made use of the readily available *skimage* library. Using the *regionprops* function, which finds regions containing patterns. By using images such as Figure 2 as arguments, the only recognized patterns of area greater than 1 were the numbers found in the image.

Fig. 1. Raw, unprocessed image taken from the training data.

Fig. 2. The result of removing all background noise from Figure 1 using the method described above.

We used the data provided by the function to draw the smallest possible bounding square around each digit, and selected the digit contained inside the square of greatest area. Finally, using the *transform.resize* function, also included in *skimage*, we resized only that digit to a 28x28 grid. The result is shown below:

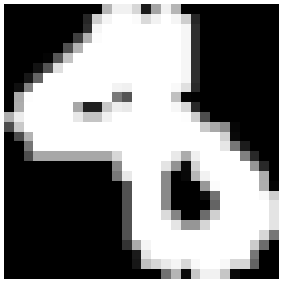

Fig. 3. 28x28 expansion of the largest digit in the original image, as determined by our preprocessing algorithm.

The decision to preserve that many features for our final input vectors was based upon trials on smaller numbers of features. Also, since our digits were already slightly pixelized in the original images, key features of certain digits were being attenuated as we downsized. Take Figure 3 as an example. The top hole of our 8 is barely bigger than a 5 pixels (out of 784, so approximately 0.006% of our final image). At smaller scale, that hole, which is a constituting element of an 8, would have been reduced, until potentially being made turned into active pixels by its surroundings. Therefore, the decision to train our models using a large amount of features was to help reduce the possibility of such ambiguous data pairs throwing our models off.

III. Algorithm

A. Linear SVM

The linear SVM library from Scikit-learn package was used to test the performance of SVM on the given dataset. Given n-dimensional $(x_1, \cdot x_n)$, the linear SVM kernel separates the classes according to geometric distance. Closer input points will fall under the same class while further input points will tend to fall into different classes.

B. Feed forward Neural Network

For the feed forward neural network, one hidden layer was used with 10 output nodes to determine which of the 10 digits the input corresponded to. Each node represents a perceptron where the outputs of the layer preceding to the node is operated under dot product then the sum is transferred after being put under sigmoid function. The weights are updated by back propagation, which calculates the gradient of the output respect to weight. After back propagation, the weights corresponding to minimum squared error is calculated incrementally by adding to the weight. The rate of convergence to the minimum is determined by the learning rate η.

C. Convolutional Neural Network

The Convolutional Neural Network build on top of regular Neural Network. In our algorithm, we used 4 layer Convolutional Neural Network. In each convolutional layer the following operation is applied to its input. The input was padded around the rim of the input, and then filter was applied by dotting the portion of the input to the filter. Then the filter moves across the input with hyperparameter stride. After filtering, batch normalization is applied for data processing, and then ReLU activation is used. This is applied twice before the pooling layer.Then the weights are updated again using backpropagation.

IV. Methodology

In splitting the training set and validation set, we randomly selected 40000 pictures out of 50000 as training set and chose the rest as validation set.

For the Linear SVC model, we selected 10000 pictures randomly out of our training set to train the model due to limited resources and time. The following hyperparameters and their values were used

1) Penalty Parameter $C : \{0.01i | i \in \{1, \cdots, 1000\}\}$
2) Maximum iteration$M : \{1000, \cdots, 10000\}$
3) Tolerance $\epsilon : \{0.01i | i \in \{1, \cdots, 1000\}\}$

For the feed forward neural network, one hidden layer was used. Same split of validation set and training set was used. The hyperparameters are

1) number of nodes in hidden layer $N : \{4, \cdots 50\}$
2) Learning rate $\eta : \{0.1^{i-6} | i \in \{0, \cdots, 6\}\}$
3) Tolerance $\epsilon : \{0.1^i | i \in 1, \cdots, 5\}$
4) Maximum iteration $M : \{1000, \cdots, 10000\}$

Now for the convolutional neural network, the number of hyperparameters that was tried is lower. This is due to limitation in computational resources and time. With 4 different convolutional layer, 4 ReLU layer and 2 pooling layer. The topology of the convolutional neural network was the same throughout the experiments. With the given topology following hyperparameters were tested:

1) Learning rate $\eta : \{0.01^{i-4} | i \in \{0, \cdots, 10\}\}$
2) Tolerance $\epsilon : \{0.01i | i \in 1, \cdots, 1000\}$
3) Maximum iteration $M : \{1000, \cdots, 10000\}$
4) Number of epochs $E : \{1, \cdots, 50\}$
5) Batch size: 50,100,150

In determining the best hyperparameters, the F1 score performance was used.

V. Results

The following results were given for neural network. In picking the hyper-parameters, hyperparameters were independently picked in attempt to reduce computation time. When controlling for hyperparameter other hyperparameters were set to lowest value. For greater finesse in tuning however, it is recommended that hyperparemeter to picked conditionally on all other parameters. Below is the table of result indicating performance of our models. The F-score was calculated on validation set.

A. Linear SVC

Using all optimal hyperparameters we get around 0.79 F-score.

TABLE I
CONTROLLING FOR PENALTY PARAMETER C

Penalty parameter (C)	F- score
0.01	0.234
0.05	0.323
0.1	0.332
0.2	0.423
0.4	0.523
0.8	0.567
1.6	0.765
3.2	0.656
6.4	0.766
10	0.542

TABLE II
CONTROLLING FOR MAXIMUM ITERATION

Learning rate (η)	F- score
1	0.234
0.1	0.452
0.01	0.546
0.001	0.677
0.0001	0.754
0.00001	0.766

TABLE III
CONTROLLING FOR TOLERANCE

Tolerance (ϵ)	F- score
0.1	0.255
0.01	0.256
0.001	0.275
0.0001	0.269
0.00001	0.311

B. Neural Network

After using the optimal values for each parameters, the F-score reaches around 0.91

TABLE IV
CONTROLLING FOR NUMBER OF NEURONS IN HIDDEN LAYER

Number of nodes	F- score
40	0.723
60	0.745
100	0.782
200	0.811
300	0.824
400	0.849
500	0.848
1000	0.849

TABLE V
CONTROLLING FOR LEARNING RATE

Learning rate (η)	F- score
1	0.234
0.1	0.452
0.01	0.734
0.001	0.832
0.0001	0.847
0.00001	0.849

TABLE VI
CONTROLLING FOR TOLERANCE

Tolerance (ϵ)	F- score
0.1	0.723
0.01	0.789
0.001	0.817
0.0001	0.846
0.00001	0.850

C. Convolutional Neural Network

Results for Convolutional Neural Network was harder to find due to computational time taken for each training. Some value for hyperparameters, often accurate ones, will required around 3 - 4 hours even with GPU(s).

TABLE VII
CONTROLLING FOR NUMBER OF NEURONS IN HIDDEN LAYER

Number of epochs	F- score
40	0.902
60	0.912
100	0.914
200	0.932
300	0.936
400	0.936
500	0.937
1000	0.952

TABLE VIII
CONTROLLING FOR LEARNING RATE

Learning rate (η)	F- score
1	0.764
0.1	0.462
0.01	0.754
0.001	0.862
0.0001	0.847
0.00001	0.899

TABLE IX
CONTROLLING FOR TOLERANCE

Tolerance (ϵ)	F- score
0.1	0.723
0.01	0.789
0.001	0.817
0.0001	0.846
0.00001	?

TABLE X
CONTROLLING FOR BATCH SIZE

Tolerance (ϵ)	F- score
50	0.723
100	0.789
150	0.817

With the optimum hyperparameters we get the following

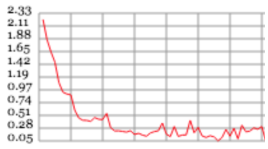

Fig. 4. Running CNN with best set of hyperparameters

which is around 0.97 to 0.98 after 13 hours of training.

VI. Discussion

Our methodology was tuned for decreasing the amount of time. In doing so, the performance of our models were not the best as it could have been. Except for the Convolutional Neural Network, we spent the minimum amount of time possible to get the result. The implementation of feed forward neural network require most amount of improvement to be used practically. In storing the weights and the intermediate calculations, simple list was used as a data structure. In small number, this was sufficient enough to converge to satisfactory result. However, in bigger datasets, training often was never completed as the list storing the weights and gradients became larger. For Convolutional Neural Network, we referred to studies already done on original MNIST dataset. Many models and studies were done previously by others, and their contributions helped tremendously in guiding towards making a better model. Greatest advantage of our approach is that we managed to accurately prep-process our data in the form that resembles original MNIST dataset. With this we were able to refer to what others have done previously on MNIST dataset in classification. However, such data processing, took substantial amount of time, and hyperparameters had to be chosen independently. Further work could be done on optimizing for time with sacrificing minimum accuracy.

Statement of Contributions

Preprocessing the Data was our initial task, We exchanged ideas and in the end we came up with the solution described in the introduction. Nicolas Fortier wrote the small code snippet we then ran our data through. As for the models, Peter Park took the lead in the development and tuning of both Neural Network models. Nicolas Fortier provided general support in both Neural Networks' (Convolutional and Feed forward) development, as well as helping with the tuning of the various hyperparameters. As for Yassine Mhedhbi wrote the code for the linear SVC using a built-in library while tuning the different hyperparameters and helping with other models. The choice to develop a Convolutional Neural Network was made as a team, after going over academic literature outlining various machine learning models' performances tasked to recognize digits.

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

VII. Appendix

Other methods for creating training data was also attempted. Here we describe couple methods included in our models in attempt to increase our perforrmance. These methods were already tried in regular MNIST studies.

A. Random Rotation

Random Rotation to up to $45°$ clockwise and counter clockwise. This is after the noise in the background is cleared. Here is an example of what random rotation would do to the input.

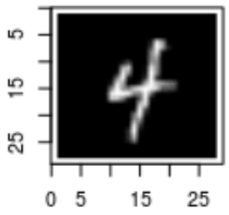

Fig. 5. Original image

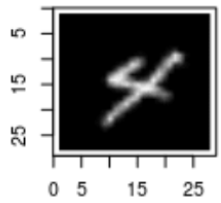

Fig. 6. Rotation

B. Random Translation

Similarly Random Horizontal/vertical translation was applied uniformly up to 3 pixels. This however seemed unnecessary as initial capture of the digit occupying the largest area would already center the 28 x 28 image in the test/validation set.

All these methods were used after initial data processing where the images were condensed from 64 x 64 to 28 x 28. Main purpose of having such data process was to decrease computation time and to take advantage of other studies which seem to have used 28 x 28 images more frequently. Quantifying improvements due to using such methods were not measured in our project due to limited time.