Digit Recognition

Kaggle Team Name: getsetgo

Jeremy Georges-Filteau
McGill University
260713547
jeremy.georges-filteau
@mail.mcgill.ca

Ramchalam Kinattinkara Ramakrishnan
McGill University
260711189
ramchalam.kinattinkararamakrishn
@mail.mcgill.ca

Sayantan Datta
McGill University
260670123
sayantan.datta
@mail.mcgill.ca

1. INTRODUCTION

Presented with a set of images containing two handwritten digits, our task was to implement and test different machine learning approaches to classify these images based on the sum of both digits they contain. A corpus of 100k images was provided along with the labeled sum of digits to train our model. A testing subset of the data is the basis of a competition on the Kaggle platform to compare the performance of our trained models with other teams. We tested and evaluated a logistic regression model, a fully connected feed-forward neural network and multiple architectural variations of a convolution neural network.

2. RELATED WORK

The problem of digit classification has been extensively studied in the past. Many researchers concentrate on the task of feature extraction [3]. Geometric or topological feature extraction has been used successfully [2]. We experimented with this type of feature extraction.

Convolutional neural networks (CNN) have been around since 1982, but it was only after the advent of GPUs that they came to the forefront of all other image recognition techniques. CNNs are, in essence, a variation of neural networks inspired by the visual cortex of rats. They are, however, computationally demanding and it is not feasible to train them without GPUs. CNNs have been used successfully in digit classification with an accuracy of over 99 %.

A simple CNN architecture essentially consists of alternating convolution and pooling layers. The former computes the output of neurons that are connected to local regions in the input space. The latter will perform a downsampling operation along the spatial dimensions . Finally, a fully-connected layer (FCL) will compute the class scores, resulting in a one-dimensional output ([1x1x19] for the 19 classes in our specific example). In this way, CNNs progressively reduce the dimensionality of the input pixel values to the final class scores. Many traditional image processing techniques have been used in conjunction with machine learning algorithms such as SVM, kNN and neural networks [4], but so far none of them are able to match the performance of CNNs on complex datasets.

3. PROBLEM REPRESENTATION

3.1. Image denoising

The input provided to us contained lots of noise. Every pixel with a value less than 255 was set to black. All our training and test data were denoised and the same was used by all three models. It is almost always a good idea to perform some scaling of input values when using neural network models. So in the case of CNN, post denoising the data, the input was normalized (divided by 255) to get values between 0 and 1

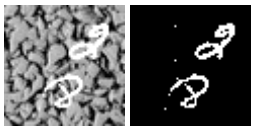

Figure 1: Image denoising

4. ALGORITHIM SELECTION, IMPLEMENTATION AND RESULTS

4.1. Logistic Regression

We used logistic regression as our baseline classifier for comparison against more advanced classifiers described in subsequent sections.

Since we are dealing with 60x60 images, our feature space is a vector of size 3600. For this purpose, we simply flattened our image into vectors of width 3600. After cleaning, the feature matrix was mostly sparse as only a small fraction of pixels were non-zero. Normalization of features is not necessary as all our feature have the same scale, i.e

from 0 to 255. Our labeled data consist of 19 classes which correspond to the sum of digits in the image. Since logistic regression is a binary classifier, we used one-vs-all classification to train our model. In the one-vs-all method, we train 19 different classifier with a real-valued dependent variable as output and the class with highest output value is selected. The dependent variable is set as positive if it belongs to the class being trained and negative otherwise. For this purpose, we used Scikit learn library which automatically does the task of multi-class classification when the multi-class option is set to 'ovr' or one-vs-rest.

4.1.1 Training Logistic Regression

We did 70-30 split of the training data for measuring training and cross validation performance. A 2-fold cross validation was then used. The reason for selecting 2-fold cross validation is purely due to limited computational capacity. We used SAG[7] or Stochastic Average Gradient solver for minimizing our loss function. SAG was chosen as it provides faster[8] convergence compared to other solvers. Each SAG iteration computes gradient for one random example and borrows the rest of the gradient values from previous iteration. Thus cost of one iteration is independent of number of examples. SAG iteration takes the form:

$$x^{k+1} = x^k - \frac{a_k}{n} \sum_{i}^{n} y_i^k$$

where at each iteration a random index $i_k$is selected and we set

$$y_i^k = \begin{cases} f_i^{'}(x^k), \text{ if i = } i_k \\ y_i^{k-1}, otherwise \end{cases}$$

We used L2 regularization as it is the only regularization compatible with SAG for our Scikit learn library. We tested two values regularization parameter, 1 and 1e4 to measure our training and cross-validation performance. Also, we did not try to increase our regularization incrementally in smaller steps as we first wanted to see the impact of regularization before fine tuning it. As can be seen from our result, increasing lambda reduces our validation accuracy negligibly. Maximum iteration of 500 was selected as increasing iteration count further was not computationally feasible for such large data-set using our laptops. However, better accuracy could have been achieved with higher iteration count.

4.1.2 Logistic Regression Results

Our training set accuracy is 7.71%, and our cross-validation set accuracy is 7.66% with regularization set to 1. Increasing regularization to 1e4 decreased our accuracy further to 7.69% and 7.64% for training and cross validation.

4.2. Feed Forward Neural Network

A fully connected feed-forward neural network with a sigmoid activation function was implemented as a second baseline method. The network was trained using a standard back-propagation procedure on two different set of features obtained from the data. In both cases, the data was first processed to eliminate noise. The categorical output was first transformed using one-hot encoding. For the first set of features, the raw pixel data was fed into the model (m=3600). For the second set of features, a geometric region based approach was trialed. Regions were determined in the images using the skimage.measure library. The two largest regions obtained for each image were then analyzed with skimage.measure.regionprops to produce a set of measurements (m=14) such as: area, perimeter, convex area, solidity, eccentricity, equivalent diameter, Euler number, extent, filled area, etc. An example of two regions is presented in Figure 2, where white and Grey pixels are assigned to each region.

Figure 2: Example case of splitting an image into regions (white and grey pixels)

4.2.1 Training Feed Forward Neural Network

As its basic structure, the neural network was comprised of 3600 or 14 nodes in the input layer, hidden layers of various quantity and sizes and 19 nodes in the output layer. Multiple conformations were tested for the hidden layers such as: 2 layers of 50, 100, 200 nodes or 10 layers of 10, 20 nodes. Solving was done using stochastic gradient descent for a set number of iterations or a threshold on the loss function. Multiple values were also tested for the learning rate (Figure 9). Due to computational constraints pertaining to our implementation not being optimized, we were limited to training the neural network on 10000 entries of the dataset.

4.2.2 Feed Forward Neural Network Results

Unsurprisingly the features taken from the raw pixel data yielded results that are indistinguishable from the random baseline, with an accuracy varying between 6% and 15%. At first glance this seems better than random, however, the model tends to over-predict values that are more common in the training set and the results are wildly inconsistent between each trial (even for a large number of iterations ¿ 500). Given the sum of two random digits, the distribution will favor values around 8, 9 or 10. The model achieves

slightly better than random results by fitting to this bias. None of the variations on the hyper-parameters yielded significant improvement and cross-validation was considered unnecessary. It was however observed that lower values of the learning rate tended to make the loss function converge at a lower more stable value. The same observation was made for a 2 hidden layer architecture with more than 100 nodes. This improvement on the stability of the loss function did not reflect on the results for the accuracy.

For the geometric features, the results showed less variation between each run and hovered around 9% accuracy. Again, a lower learning rate made the loss function converge to a lower value but made no considerable difference in the accuracy. The number of hidden nodes or layers made no difference.

To confirm that our implementation of a neural network was functioning correctly we also tested both set of features on the scikit-learn implementation of a neural network with the same parameters, yielding similar results. Our implementation was also tested on other data-sets such as the well know Iris data-set and the Poker data-set from the ICU Machine learning repository. The results confirmed that our neural network did not contain programming errors since accuracies obtained were above 80%.

4.3. Convolution Neural Network(CNN)

Convolutional Neural Networks are said to model animal visual perception and are one of the most widely used methods applied to visual recognition tasks. We have used the Keras library to implement a CNN. It is a high level neural networks library to be used on top of Tensor-flow. It was used efficiently for the successful implementation of the famed MNIST data-set with an accuracy close to 99%. The raw data is provided as images with 60*60 pixels. The features for our algorithm was, in effect, 3600 different pixels representing different points in the image. So essentially we had 100k as the number of training examples and each of the 3600 pixels as features. The data cleaning process was exactly the same as mentioned previously. Every non-white pixel (¡255) has been made 0. We went by the notion - less noise the better. Whether this would improve the performance significantly is to be seen but this would certainly not degrade the performance of the CNN in its prediction. As per the standard MNIST implementation, we also normalized the data dividing by 255 to get a value between 0 and 1. For the output, Y, we used one-hot encoding to convert to binary and represent each class as a combination of 0s and 1s. This is required for the CNN model to function.

4.3.1 Training CNN

Our first approach was to emulate the successful MNIST single digit architectural implementation. We wanted to use this as a benchmark to improve upon. The successful basic LENET implementation included 2 convolution layers, 2 pooling layers, a dropout layer and 2 dense (fully connected) layers [1]. The MNIST data, however, is free of noise, centered, normalized and aligned with 28x28 pixels. We thus knew this would not give us the ideal result but we used this as a base for our first experiment. We opted for a standard sequential model in Keras, which is essentially a linear stack of layers. Our implementation of this model for our data-set gave us an accuracy of around 80% after training for 10 epochs. Using a 70-30 training test split, we then further experimented with 3 different variations on the number of layers for this model which are further described in the following sections.

6-Layer Architecture with constant filter map In this first architectural experimentation we used 8 hidden layers (6 Convolution and 2 feed forward). The input, X, which was in the form 60x60x1, was fed into the first convolution layer. A *convolution2D* model was used for filtering varying windows of 2-dimensional input. We used 64 filter maps and a relu activation function throughout all our convolution layers. A Max-pooling of 2x2 was used in all pooling layers as this was the standard in the successful implementation of the MNIST data-set.

We first used a high window size of 7x7 in the first input layer with a stride of 1, reducing the 60x60 initial data dimensions to 54x54 (x64 filter maps), followed by a layer with window size 5x5, again reducing the dimensions to 50x50. Following this, a pooling layer was added which further reduces the size of the data to 25x25. We then repeated this pattern with a further 2 convolution2d layers of window frames 5x5 and 3x3 respectively, reducing the dimensions to 19x19, followed by another pooling layer reducing to 9x9 (x64 filter maps).

This was then fed into a dropout layer with probability 20, which was as per the standard implementation. Before feeding the fully connected layers, the data was flattened. Two fully connected dense layers were used with 128 and 50 neurons respectively. Finally, this was fed into the output layer with 19 nodes, each representing a different class.

We achieved a maximum testing accuracy of 91% after training for 20 epochs with a batch size of 200.

9-Layer Architecture with varying filter maps In this method, we used a slightly different approach with 11 hidden layers (9 convolution and 2 fully connected). We varied the number of filter maps across the layers. We used 2 convolution layers followed by one pool layer as our base model. We connected 3 such models to create a 9 layer architecture. Our logic was that since we had twice the number of digits as compared to the original MNIST data-set, a larger number of filter maps would help the network use

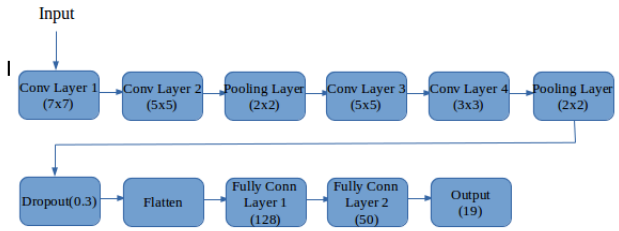

Figure 3: 6 Layer Model architecture

different permutations. We used 32 filter maps for first 2 convolution layers, 64 for next 2 layers and 128 for last 2 convolution layer. Max-pooling layers with 2x2 frames were also added. This improved the accuracy slightly but it was computationally more costly. The final testing accuracy for 20 epochs was around 94.25%.

11-Layer Architecture with constant filter map In this architecture, we went by the logic that as the number of hidden layers increases the accuracy tends to increase, up to a certain extent. So here we used a 11 layer architecture. We reverted to 64 filter maps (due to computational constraints with the additional hidden layers) of frame size 3x3 for all the convolution layers. A Max-pooling of 2x2 was also used. The dropout was increased from 0.2 to 0.3 over the fear that we might over-fit the data. The activation function used was relu with the loss function as Categorical cross-entropy and adam optimizer (computationally much faster than the standard sgd [5]). We used L2 regularization for the fully connected layers. It was computationally not realistic to compute the regularization hyper-parameter using k fold cross validation. We thus systematically tested on a few values and took the best value of 0.01. This 11 layer architecture gave the best results for us, by far. We achieved a validation accuracy of 97.04% after training for 15 epochs. Each epoch took around 1500 seconds on a GPU (NVIDIA GEFORCE GT 240M).

4.3.2 CNN Results and cross-validation

For each of the tested models, we got different accuracies with the best accuracy for the 11 layer architecture (Figure 5). We used 3 fold CV on our 11 layer architecture to check for over-fitting on the number of epochs. We settled for k=3 folds considering the computational demand of such a test. We ran it for close to 2 full days on the GPU, saving the intermittent results. The cross-validation accuracy plateaus at around epoch = 15. We thus used epoch = 15 for our final submission. Moreover, as the epoch exceeds 15, the accuracy was oscillating. Hence it seemed futile to go further and probably over-fit the data. We also tested the effect

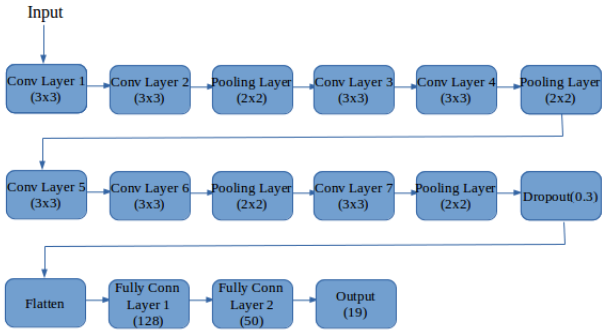

Figure 4: 11 Layer Model architecture

Model	Training Accuracy	Testing Accuracy	Average Time per epoch(sec)
6-Layer	98.5	91.2	550
9-Layer	98.45	95.34	1108
11-Layer	97.36	97.04	1489

Figure 5: Training and testing accuracy for the different model architectures of the convulotional neural network

of the number of hidden layers in the model on the accuracy (Figure7). For the architecture we followed, it can be seen that as the number of layers in the model increases, the accuracy also tends to increase.

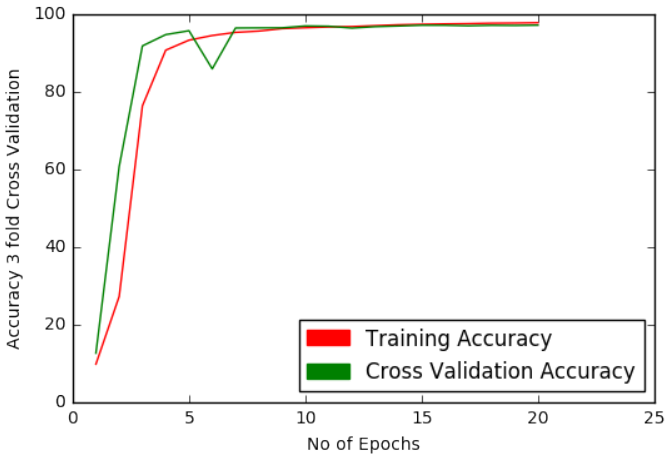

Figure 6: Accuracy for the training and validation set versus number of epochs ran in the 11 Layer Architecture with 3-fold cross-validation

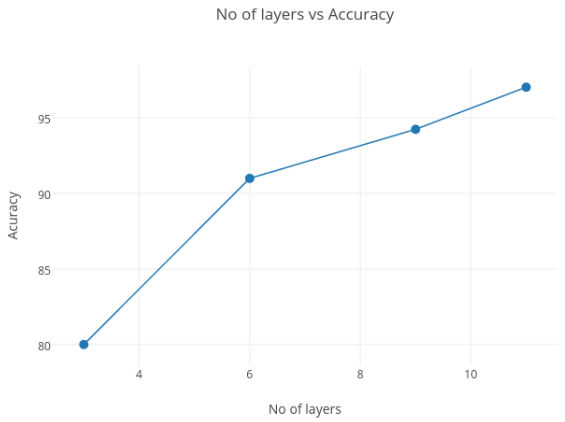

Figure 7: Accuracy versus number of hidden layers in the CNN (6, 9 and 11)

5. DISCUSSION

The primary motivation behind the inclusion of logistic regression as a classifier was to obtain a strict baseline for our more advanced classifiers. Clearly, Logistic Regression being a linear classifier is not suitable for classifying data with complex non-linear decision boundaries. While logistic regression might have performed better with more advanced image processing techniques applied to the image but then that would beat the purpose of using logistic regression as a baseline classifier.

The results obtained for the feed-forward neural network are not unexpected, considering the digits in the images are randomly placed and oriented. It is intuitively impossible for a neural network to learn anything from such data. Some pre-processing could, however, improve the results. The digits could be first separated, centered and rotated to a standard position. Classifying the digits separately and summing them would then be trivial. However, using a convolutional neural network, more adapted to such data, is a simpler approach.

The intuition behind the geometric approach was that even though the digits are handwritten, they would share similar geometric properties. One can suppose, for example, that the ratio of area to perimeter of a digit is specific. However, a problem arose with the separation of the images into regions. While a large proportion could be separated into 2 neatly defined regions corresponding to the 2 digits, many yielded 3 or more regions. In the end, it did not seem that the data could be classified based on these features.

As expected CNN performs better than the standard feed-forward neural network when it comes to image classification. There is always a trade-off between the computational cost and accuracy. We used NVIDIA GEFORCE 240M GPU to train the CNN model. Nevertheless, it took approximately 5 hours for 10 epochs of the 11-Layer Architecture.

The fact that CNN takes the input and parses it frame by frame as per the appropriate dimensions, makes it ideal for image recognition. In our case specifically, when the digits have no particular alignment, this approach works the best. We feel that with even better computational capacity, we could bump up the accuracy even further, notably having the capacity to run cross-validation on every parameter.

Unfortunately, considering that our model is too computationally heavy to be cross-validated on every possible parameter, we had to systematically estimate some of the values. There are multiple hyper-parameters used in CNN which could benefit from being refined using k-fold Cross Validation. Currently, we have done 3 fold cross validation only for the optimum number of epochs vs accuracy and found that the optimal quantity is 15 epochs 6. We believe optimizing all the hyper parameters using cross-validation could pay greater dividends.

5.1. Alternate Approaches

We also attempted segmentation of digits using OpenCV bounding boxes[9] and feeding the segmented digits through our classifier trained on standard MNIST data. For this purpose, we first segmented the images into separate digits. Each digit was centered and re-scaled to 28x28 so that it can be recognized by our standard MNIST classifier. Our single digit MNIST classifier was trained on a two layer feed forward neural network with 850 and 50 neurons in the hidden layer. We used 55000 out of 60000 labeled MNIST digits to train our single digit classifier. Cross-validation accuracy of our single digit classifier on standard MNIST data was 97.2 percent. For our final prediction, i.e. the sum of digits, we add two of our predicted class to produce our final output. This final classifier scored 60.6 % on our cross-validation data which is our entire dataset containing 100000 training examples. Primary reason for relatively low accuracy compared to other models is due to the fact that we were unable to segregate all the digits using bounding boxes.

6. STATEMENT OF CONTRIBUTIONS

We hereby state that all the work presented in this report is that of the authors. Each author played a role in implementing a specific classifier out of the three. SD implemented the Logistic-Regression model and image segmentation model whilst JGF implemented the Feed Forward NN algorithm and geometric feature pre-processing; RR worked on the predictions via CNN classifier; all authors contributed to the approach, design of the feature sets and the written report.

References

[1] Learning Algorithms for Classifications
http://yann.lecun.com/exdb/publis/pdf/lecun-95a.pdf

[2] New efficient algorithm for recognizing handwritten Hindi digits
PROCEEDINGS OF THE SOCIETY OF PHOTO-OPTICAL INSTRUMENTATION ENGINEERS (SPIE), 467068-73, SPIE-INT SOC OPTICAL, ENGINEERINGBELLINGHAM2002, 0277-786X, 0-8194-4410-3

[3] Arabic handwritten digit recognition,
International Journal on Document Analysis and Recognition, Article vol. 11, no. 3, pp. 127-141, Dec 2008.

[4] Handwritten Digit Recognition Based on Principal Component Analysis and Support Vector Machines
Communications in Computer and Information Science, 214595-599, Springer-Verlag GmbH Berlin HeidelbergBERLIN2011, 1865-0929, 9783642227257

[5] ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION
https://arxiv.org/pdf/1412.6980.pdf

[6] MDig: Multi-digit Recognition using Convolutional Nerual Network on Mobile
http://web.stanford.edu/class/cs231m/projects/final-report-yang-pu.pdf

[7] Mark Schmidt, Nicolas Le Roux, Francis Bach. *Minimizing Finite Sums with the Stochastic Average Gradient - Page 4*
http://hal.inria.fr/hal-00860051/document

[8] Sckit-Learn solver comparison.
http://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_comparison.html

[9] OpenCV: Contours and Bounding Boxes
http://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html

[10] Keras Documentation
https://keras.io/

[11] CNN Basic Architecture
http://cs231n.github.io/convolutional-networks/

7. APPENDIX

Class	Precision	Recall	F1 Score
0	0.95	0.96	0.96
1	1.00	0.98	0.99
2	0.97	0.99	0.98
3	0.99	0.97	0.98
4	0.97	0.98	0.97
5	0.97	0.97	0.97
6	0.96	0.98	0.97
7	0.97	0.98	0.97
8	0.97	0.97	0.97
9	0.97	0.97	0.97
10	0.98	0.96	0.97
11	0.98	0.97	0.97
12	0.98	0.96	0.97
13	0.97	0.97	0.97
14	0.97	0.96	0.97
15	0.97	0.97	0.97
16	0.94	0.98	0.96
17	0.96	0.96	0.96
18	0.92	0.96	0.94
Average	0.97	0.97	0.97

Figure 8: The precision and recall values for the 11-Layer Architecture.

Parameter	Values
Learning rate	0.0001 to 10
Number of itterations	100 to 1000
Number of hidden layers	2 to 10
Hidden layer sizes	2 to 200

Figure 9: Various hyper-parameters tested for the feed-forward neural network.