# Machine Learning Digit Classification

Joanna Halpern
joanna.halpern@mail.mcgill.ca
260410826

Talia Wise
talia.wise@mail.mcgill.ca
260659717

Chenghao Liu
chenghao.liu@mail.mcgill.ca
260736137

***Abstract*—This paper compares the performance of three machine learning algorithms on a dataset of hand-written digits. The problem is complicated by the challenge of detecting the largest digit in each image. We consider a Linear SVM, a Neural Network and a Convolutional Neural Network, and find that the CNN produced the best classification accuracy of over 93 percent on our testing data.**

## I. Introduction

Machine learning has been gaining increasing attention as a powerful class of methods that allow for the realization of many important real-world applications by making accurate predictions based on previous data (Goodfellow et al. 2006). Pattern recognition, such as speech recognition and image recognition, is an important machine learning application, especially due to its potential real-world uses, such as face detection and computer-aided diagnosis (Goodfellow et al. 2006). Optical character recognition is one type of machine learning pattern recognition which has proved very effective. In this project, we aim to detect and classify hand-written digits using machine learning techniques. The dataset consists of images with several hand written digits of varying sizes pasted onto messy backgrounds. The goal of the project is to detect and classify the largest digit in each image correctly.

Our approach was to first construct a feature extraction module which finds the largest digit in each image. We then compared classification results of three different machine learning algorithms, a linear SVM, a neural network and a convolutional Neural Network on the extracted features, with extensive hyper-parameter tuning. The linear SVM serves as a baseline in our study, while the fully connected neural network reveal the potential of readily-implemented machine algorithms, and at last, the convolutional neural network present itself as a readily-available algorithm that produces satisfactory and directly applicable results. Overall, we found that the Convolution Neural Network produced the best results with an accuracy of 0.935 on the Kaggle test data. The Neural Network classified the data significantly less well, with a validation accuracy of 0.867 and the Linear SVM followed with a validation accuracy of 0.779.

## II. Feature Design

For preprocessing, the largest digit was first identified and then extracted from each image. In order to optimally separate the digits from the image backgrounds, each image was converted into a binary format where a pixel value of 255 in the original image corresponds to a value of 1, and anything less than 255 was converted to 0. While this conversion does slightly alter the shape and size of some digits, we found that restricting foreground (i.e. digit area) to pixels with a value of 255 appears to be the only method to ensure that all the digits are extracted correctly without being transfigured by background elements. Any lower cut-off point for binarization was found to distort the shape of some of the digits when the images where converted into binary format. This distortion also caused more incorrect digits to be selected as the largest digit since it increased the size of the bounding box of that labeled region.

We also experimented with several different image thresholding and background detection algorithms from openCV, such as otsu-threshold and triangle threshold, but found that while some of these algorithms did improve background detection on most of the images, none of the filters worked on all of the images. Because of this we chose to use the simple binary thresholding described in the paragraph above.

In order to detect the largest digit we first found each digit's bounding box using the skimage regionprops method. We then compared the pixel area of each digit, the area of each regions bounding box and the maximum of height and width of each regions bounding box (which is equivalent to finding the area of the bounding square). We found that the maximum region length (maximum bounding square) worked significantly better than either of the other two options tested, although it still did not produce perfect results. This method likely worked best because the dataset was created by resizing digits of the same height and occasionally rotating them, so finding the digit with maximum height or width returns the digit which was enlarged the most in each image.

We then resized the bounding box of the largest digit to a 15 by 15 pixel image and saved this to an array. We tried adding a border of 1 background pixel around each image, but found that this did not improve results.

We similarly extracted the bounding box surrounding each digit in the MNIST dataset from keras and resized the image to 15 by 15 in order to amplify our training dataset.

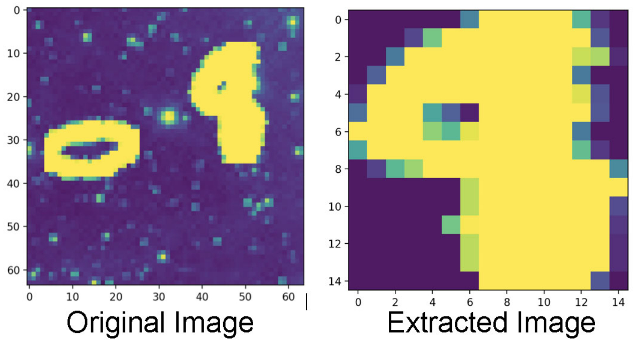


Fig. 1. Feature extraction

## III. Algorithms

### A. Baseline Linear Learner

We first trained a Linear Support Vector Machine on the data in order to get a baseline validation accuracy to compare with our other results. We used the LinearSVC implementation from scikit-learn. Linear SVMs are supervised learning models that find hyperplanes to best maximize the separation between the different classes of data in space. In this case the SVM takes in each digit image as a one dimensional array where each pixel is a feature.

### B. Neural Network

A simple, fully connected, feed-forward artificial neural network was constructed from only numpy, and trained using back propagation to illustrate the power of relatively easily-built algorithms in the field of image recognition.

An artificial neural network is a collection of nodes organized into layers, with the first layer consisting of nodes made up by the input of the data, the last layer consisting of nodes that represent the final desired output, and hidden layers which connect themselves to the first, last, or other hidden layers, which bear the learning ability in the algorithm. Much like the biological neural network, in our system, every connection between the nodes, or synapses between neurons, is allowed to transmit information from one neuron to another. Once the information is transmitted to the next neuron via the synapse, the next neuron may process it and pass it on forward. In our implementation, the neurons are connected by a real number, or weight, and each neuron outputs the result of a non-linear function of the sum of the inputs, the sigmoid function in our case.

Overall, the neural network is trained by backpropagation. This involves propagating forward through the network first using some randomly generated weights, calculating the error of the network, and then propagating the output back through the network, using training pattern targets, to generate the differences between the targets and the actual outputs of all output and hidden neurons. After this, the gradient of the weight is found by multiplying the output delta (i.e. the difference we mentioned earlier) and the input activation; at last, some parts of the weight gradient, determined by the learning rate, is subtracted from the initial weight to result in a new gradient. This then is performed over many epochs to find the optimal accuracy, tested via a validation dataset. Similar procedures are performed over different number of mini-batches that are sampled into the network, different learning rates, and different number of hidden layer neurons to tune these hyper-parameters. Note that theoretically, there are possibilities to implement more than one hidden layer, although this was not performed both due to limited possibility for improvement, and significantly better results received from the convolutional neural network below.

### C. Convolutional Neural Network

We chose to use a Convolutional Neural Network for our third algorithm because CNN's are excellent at detecting features of 2D images. Research comparing different methods of classifying handwritten digits shows that CNNs were found to obtain the most accurate results (LeCun et al. 1998). A CNN classifies images by first downsampling each image using convolutions and pooling in order to preserve the information of the images but make the data much smaller. For the convolution layer, a rectangle called a kernel (with specified length and width), goes through the image in every possible way. Each time it goes over a part of the image it applies convolution, which is done by taking the sum of the product of each element with the kernel. This is done with multiple kernels, where the number of kernels is called the depth, and each kernel has an additive bias. Finally, an activation function is applied to all of the pixels of the new image. Next the pooling layer is applied, which takes (usually disjoint) rectangles of the image and aggregates them to a single value. The method of aggregation used in the CNN is max-pooling, which returns the maximum value of each rectangle. Pooling greatly decreases the size of the image.

Next, in order to combat overfitting, a regularization technique is applied called dropout. For dropout, in each iteration, each neuron has a certain probability p of being removed. This forces the network to rely on a broader number of neurons for classification. Finally, the data is flattened to 1D and then fed into a neural network as described in the section above.

## IV. Methodology

### A. Linear SVM

For the Linear SVM we used five-fold cross validation. The hyper-parameters we tested were:

- maximum number of iterations
- dual or primal optimization
- penalty norm (l1 or l2)
- loss function (hinge or squared hinge)

- multi-class strategy
- penalty parameter of error term

We also used a version of our data preprocessing algorithm to generate an extra 30000 features and tags from the MNIST dataset. We then shuffled this data into our training set and trained the model with it as well and tested to see if this improved validation accuracy.

## *B. Neural Network*

A simple feedforward, backpropagating neural network was implemented using a flexible architecture that allows fine-tuning of the parameters involved in the neural network. In particular, the number of neurons in the hidden layer, the size of the mini batches, and the learning rate was tuned via a validation set. Note that although the architecture allows more than one hidden layer to be constructed, our testing suggested that it was not necessary as it did not improve the performance of the algorithm, and therefore is not tuned. Also note that sigmoid neurons were used in this algorithm, while the possibility remains for uses of other functions such as tanh, which we considered unnecessary as the sigmoid neurons displayed satisfactory performance for our purpose.

To explain the methodology in more details, the training dataset used was selected randomly from the overall provided training dataset with 70 percent probability, while the rest of the data formed the validation dataset. Cross-validation was not adopted because of the time required for training the neural network. The algorithm was first initialized with random weights and biases, and then trained with the training set via backpropagation, the resulting neural network was then taken and tested with the validation set via simple forward propagation.

The algorithm is optimized by tuning a variety of hyperparameters, specifically the number of neurons, the size of the batches, and the learning rate, which crucially determines the weight and biases of the final algorithm due to their mathematical significance in the training procedures. For each of the hyperparameters, a default value was assigned in the beginning and varied over three nested loops. The number of epochs were fixed through each loop to find how many epochs achieves the highest accuracy, the number of neurons was varied from 20 to 80 with 15 in each increment, the size of mini-batches was varied from 10 to 370 with 90 in each increment, and the learning rate was varied from 0.3 to 5.13 with 1.5 times difference in each loop. These values were chosen based on initial testing, and on the basis that the number of neurons and batch size are known to affect the result linearly (Goodfellow et al. 2006), we were intrigued by whether the learning rate, or its logarithm, would affect the performance linearly. Note that only the epoch number that produced the highest accuracy is recorded for each parameter setting. We should also note that, due to limited computation power and our hope for better performance from the CNN, more than 50 epochs, regularization, and entropy cost functions were not introduced, although these may have ameliorated the model. In particular, with more epochs, we may have been able to detect whether the algorithm was overfitting at low number of epochs by plotting against the cost function.

## *C. CNN*

For the third model, first a simple CNN was created and then the hyperparameters were tuned. The simple CNN first had a 2D convolution layer with the convolution depth and the size of the kernel tuned to certain hyperparameters. Next, the image went through a pooling layer with the pool size was adjusted. After that, in order to combat over-fitting, a dropout layer was applied with various dropout percentages. This was to regularize the model. Then the image was flattened to 1D so it could next be used as input to an MLP with one hidden layer. The number of neurons of the hidden layer was another hyperparameter we tuned and a rectified linear (ReLU) activation function was used. The reason for using ReLU as the type of activation function was because ReLU activations are the most popular for deep learning (https://cambridgespark.com/content/tutorials/deep-learning-for-complete-beginners-recognising-handwritten-digits/index.html). Finally, the output dense layer was applied which used a softmax activation because this is probabilistic classification. The objective to optimize in our model was the cross-entropy loss function since it is good for probabilistic tasks. The Adam optimizer was used for gradient descent.

The training data for the model was split into 80 percent training data and 20 percent validation data. Cross validation was not used because CNNs already take a long time to run and this would have made the running time much too long. The model was then fit to the training set with various batch sizes chosen as possible hyperparameters and the number of epochs was chosen depending on when the validation set got the maximum accuracy.

To tune the hyperparameters, each hyperparameter was given a default value and then the value of each hyperparameter was varied at a time. The set of hyperparameters with the highest accuracy were then chosen. The default values that were chosen were 200 for the batch size, 32 for the depth of the convolution layer, 5 for the kernel size of the convolution, 2 for the pooling size, 0.2 for the dropout percentage, and 128 for the number of neurons in the hidden layer. The number of epochs was varied each time for each hyperparameter.

First the batch size was varied given inputs 50, 100, 200, and 400. Then the depth of the convolution layer was given the inputs 16, 32, and 64. Next the kernel size of the convolution layer was given the inputs 3, 4, 5, and 6. Then the pooling size was given inputs 2, 3, and 4. The dropout percentages that were tested were .1, .2, and .4 and finally the number of neurons in the hidden layer that were tested were 64, 128, 256, and 512.

Once the hyperparameters that gave the highest accuracy of the validation set were chosen, a final model was trained with them to make a prediction for the test set. This prediction was submitted to Kaggle.

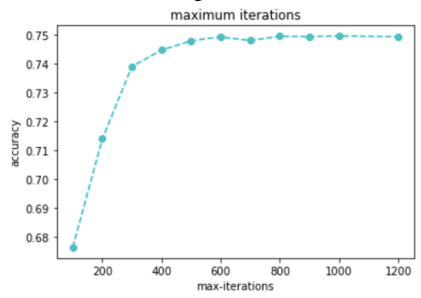
Another way that we attempted to use CNNs which did not work was to both detect which was the largest digit and then predict what that digit was. We tried to do this by first conducting the background thresholding described in the Feature Extraction section above(all pixels that were not 255 to 0). Then feeding the entire image into the CNN instead of just the largest digit. This achieved a result of only 0.33, which is about 1 in 3, which suggests that the largest digit was not being read.

## V. Results

### A. *Linear SVM*

Figure 2 shows the effect of the maximum number of iterations on training accuracy. The curve levels off at a maximum of 600 iterations so all future SVM validation models where trained at this number.

Fig. 2.



| Linear SVM | | | |
|---|---|---|---|
| loss | multi-class | MNIST | accuracy |
| **hinge** | ovr | not included | 0.735 |
| **squared_hinge** | ovr | not included | 0.749 |
| squared_hinge | **ovr** | not included | 0.749 |
| squared_hinge | **crammer_singer** | not included | 0.766 |
| squared_hinge | crammer_singer | **not included** | 0.766 |
| squared_hinge | crammer_singer | **included** | **0.779** |

Note that none of the other hyperparameters tested had any effect on validation accuracy.

### B. *Neural Network*

The simple three-layer neural network performed better than linear SVM with some edge regardless of the hyperparameters used. At the best validation performance, the neural network achieved an accuracy of 0.866702, using 80 neurons in the hidden layer, batch size of 100, 5.13 learning rate, and on the 22nd epoch out of 50. This presents a relatively large difference from the lowest validation performance of this algorithm, which was 0.755576 for 65 neurons in the hidden layer, batch size of 10, learning rate of 0.3, and on the 47th out of 50 epochs, which shows the power of hyperparameter tuning.

No direct correlation was observed for the best number of epochs with batch sizes, learning rate, or number of neurons. In general, while everything else was kept the same, as the learning rate increases, the accuracy first improves and then decreases, as depicted in an example in Fig. 3, correlating to the classical underfitting and overfitting behavior. It is also generally observed that as the batch size increases, the overall accuracy first increases and then decreases, a typical example of which is shown in Fig. 4, due to similar reasons as above. For all numbers of neurons tested, batch size of 100 showed the best overall validation performance. It is highly likely that since batch size and learning rate usually only determine the convergence rate and do not lead to overfitting, the sub-optimal results are observed mainly due to insufficient number of epochs leading to slow training.

At last, as we alter the number of neurons, the algorithm performed increasingly more accurately as the number of neurons increased, shown in Fig. 5. The performance did not show an overall decrease as neurons increased, possibly due to the fact that only 65 neurons were used at maximum because of the calculation time constrains, indicating that this hyperparameter is still in the region of underfitting and could potentially be improved further with more tuning. Although the increased accuracy with number of neurons shows a decreasing gradient, shown in Fig. 6, indicating the optimal fit is nearly achieved. This is straightforward to comprehend as the input information is inevitably cast with small numbers of hidden neurons compared to a large input of 225 pixels, leading to the phenomenon that the neuron numbers are too small for the algorithm to quickly converge and thus exhibiting a lower accuracy; on the other hand, too many hidden neurons, despite the long training time, would allow the network to memorize and thus excellently classify the training data, but not the test dataset, leading to overfitting. Note that the test dataset was not used due to limits of submissions to Kaggle.

### C. *CNN*

Table 7 shows the best accuracy achieved by using different hyperparameters as discussed in the methodology section. What is meant by best accuracy is the highest validation accuracy for however many epochs produced the highest validation accuracy for each iteration.

For batch size, in general, smaller batch sizes resulted in higher accuracy, with batch size of 50 being chosen for the test set submission. The depth of the convolution did not make a significant difference in the accuracy. A depth of

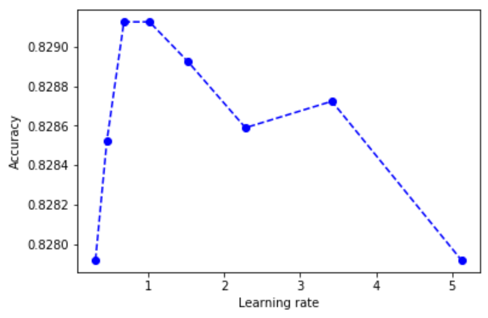


Fig. 3. Validation performance with different learning rates of a neural network of 30 neurons, with maximum 50 epochs, showing classical under/overfit behavior

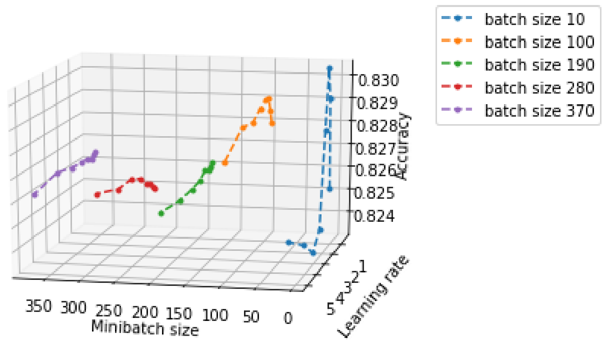


Fig. 4. Validation performance with different batch sizes and learning rates in a neural network with 35 neurons and maximum 50 epochs

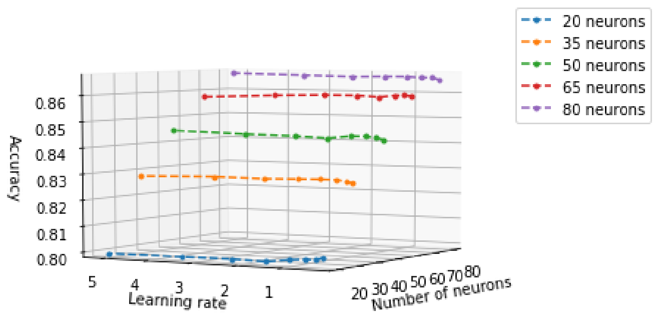


Fig. 5. Validation performance with different number of neurons with different learn rates in a neural network with batch size 100 and maximum 50 epochs

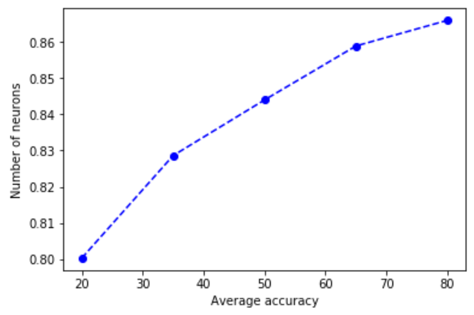


Fig. 6. Average validation performance with different number of neurons over different learn rates in a neural network with batch size 100 and maximum 50 epochs, showing a decrease in gradient and approach to optimal fit

64 was slightly better than the rest so this was chosen as the hyperparameter for the test set. The kernel size in the convolution layer also did not make much of a difference but the best result was with size 6 so this was chosen. The pooling size did make a significant difference with the smaller the size, the better the result. Therefore 2 was chosen for the pooling size. The dropout percentage did not seem to make a significant difference so a dropout percentage of 0.4 was used because it is important to combat overfitting. Finally, the number of neurons in the hidden layer of the MLP with the best result was 512. The more neurons, the better the result, but the longer the runtime.

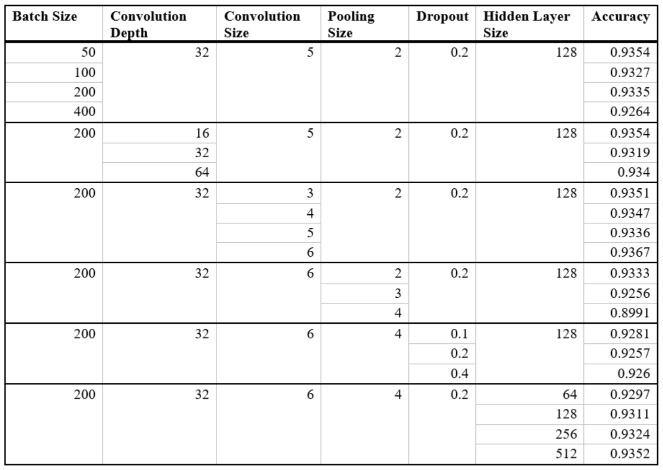
With all of these hyperparameters, 5 epochs gave the best accuracy without overfitting, as can be viewed in Figure 8. Overfitting was detected when the accuracy of the training data was higher than the accuracy of the validation data which started to occur after 4 epochs.

## VI. Discussion

Hyperparameter tuning of all the three methods delivered expected results, as usual underfitting and overfitting behaviors are observed. For Linear SVM and the simple neural network, we demonstrate that by tuning the hyperparameters, the performance of the algorithm can be improved, but the improvement is limited by the choice of the algorithm, as we see that the best performance of linear SVM is worse than that of the simple neural network, and the best performance of the simple neural network is worse than that of the CNN. This is an expected result as linear SVM, a linear classifier, has limitations on classification of potentially non-linear data, such as image pixels for recognition. Non linear classifiers like Neural Networks performs better at this task.

We found that the best accuracy came from a CNN combined with the additional MNIST training data. This model performed significantly better than either of the two others, which is in accordance with previous research on digit classification. CNN performs superior than regular neural networks for digit recognition because they first preserve the information of a 2D image through convolution layers, and then they reduce the size of the image, through pooling layers and then the image is flattened and put through a neural network. This allows the model to be trained far more efficiently than a regular neural network.

Our approach could most be improved by better feature extraction, specifically in identifying the largest digit. We found that the Convolution Neural Network detected the extracted digit correctly in every example we could find, but sometimes the wrong digit was selected as largest by the preprocessing algorithm.



| Batch Size | Convolution Depth | Convolution Size | Pooling Size | Dropout | Hidden Layer Size | Accuracy |
|---|---|---|---|---|---|---|
| 50 | 32 | 5 | 2 | 0.2 | 128 | 0.9354 |
| 100 | | | | | | 0.9327 |
| 200 | | | | | | 0.9335 |
| 400 | | | | | | 0.9264 |
| 200 | 16 | 5 | 2 | 0.2 | 128 | 0.9354 |
| | 32 | | | | | 0.9319 |
| | 64 | | | | | 0.934 |
| 200 | 32 | 3 | 2 | 0.2 | 128 | 0.9351 |
| | | 4 | | | | 0.9347 |
| | | 5 | | | | 0.9336 |
| | | 6 | | | | 0.9367 |
| 200 | 32 | 6 | 2 | 0.2 | 128 | 0.9333 |
| | | | 3 | | | 0.9256 |
| | | | 4 | | | 0.8991 |
| 200 | 32 | 6 | 4 | 0.1 | 128 | 0.9281 |
| | | | | 0.2 | | 0.9257 |
| | | | | 0.4 | | 0.926 |
| 200 | 32 | 6 | 4 | 0.2 | 64 | 0.9297 |
| | | | | | 128 | 0.9311 |
| | | | | | 256 | 0.9324 |
| | | | | | 512 | 0.9352 |

Fig. 7. Table showing all of training iterations and their corresponding validation accuracies using different hyperparameters in order to tune the hyperparameters.

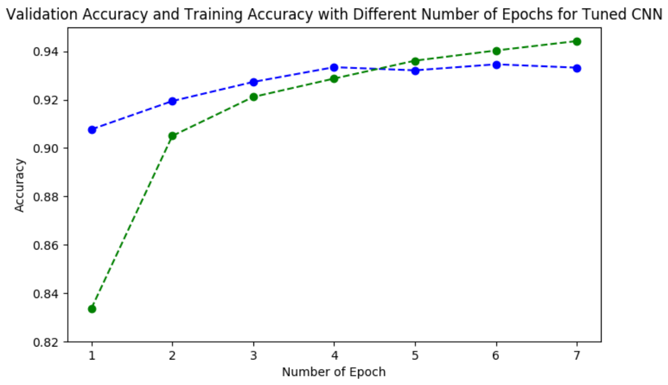


Fig. 8. The number of epochs and the model's corresponding validation and training accuracies for a model with tuned hyperparameters.

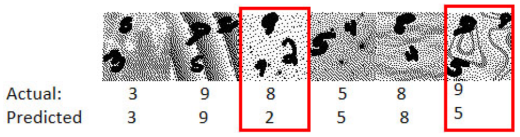


Fig. 9. Feature extraction errors

The two images in red boxes in Fig. 9 demonstrate cases where the CNN correctly identified the digit which visually appears to be the largest digit in the image, but which is not the digit tagged in the dataset. While this is partially due to the messiness of the data itself, in future work we could experiment further with feature extraction in order to correctly identify a higher percent of the largest digits. This might be done through training a CNN to identify the largest digit in each image, or through training a CNN on entire images instead of on images of extracted digits.

Additionally, further data amplification by rotating some of the extracted images and adding the rotated versions to the training data could have helped mitigate the impact of rotated digits. And the digit classification algorithm itself could also be improved by using a committee of CNNs instead of just one, as described in Ciresan et al. CVPR 2012.

The approach of creating a simple CNN and tuning the parameters worked quite well, but there are many improvements that could have been made. First of all, more layers could have been added to the CNN, including more convolution and pooling layers as well as more hidden layers. This would create a deeper network which could get better results but would also be far more complex with more hyperparameters.

For the simple CNN that was used, the depth of the convolution layer and the size of the kernels of the convolution layer, did not have a big effect. One reason for this is that all of the sizes chosen preserve close to the same amount of information about the image. Pooling size did make a difference because the preprocessed image was already so small that having too big of a pooling size would lose information.

In conclusion, we showed that by appropriately choosing the algorithm and varying the parameters of the algorithms, we can achieve a satisfactory result for handwritten digit recognition with simple architectures, the results of which not only allow us to gain deeper insights to the theories of fundamental machine learning, but also revalidate the potential of machine learning in the field of pattern recognition, if not beyond.

## VII. Statement of Contributions

All members worked together to define and understand the problem. Talia worked on feature extraction and the Linear SVM. Chenghao worked on the Neural Network. Joanna worked on the Convolutional Neural Network. All three team members collaborated to write the report.

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

## VIII. References

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

LeCun, Yann, et al. "Learning algorithms for classification: A comparison on handwritten digit recognition." Neural networks: the statistical mechanics perspective 261 (1995): 276. accessed https://pdfs.semanticscholar.org/943d/6db0c56a5f4d04a3f81db633fec7cc4fde0f.pdf

https://cambridgespark.com/content/tutorials/deep-learning-for-complete-beginners-recognising-handwritten-digits/index.html

LeCun, Yann, et al. ”Learning algorithms for classification: A comparison on handwritten digit recognition.” Neural networks: the statistical mechanics perspective 261 (1995): 276. accessed https://pdfs.semanticscholar.org/943d/6d b0c56a5f4d04a3f81db633fec7cc4fde0f.pdf

https://cambridgespark.com/content/tutorials/deep-learning-for-complete-beginners-recognising-handwritten-digits/index.htm

LeCun, Yann, Corinna Cortes, and Christopher Burges. ”MNIST dataset.” (1998).

Velikovi, Petar. https://cambridgespark.com/content/tutorials/deep-learning-for-complete-beginners-recognising-handwritten-digits/index.html

Ciresan, Dan, Ueli Meier, and Jrgen Schmidhuber. ”Multi-column deep neural networks for image classification.” Computer vision and pattern recognition (CVPR), 2012 IEEE conference on. IEEE, 2012.