COMP-551-Project4: Find the Largest Digit

Team Name: TBA, section number: 551-002
Simon Hsu, 260610820, Derek Yu, 260570997, Jason Dias 260617554

***Abstract*—Computer Vision and image processing fields are rapidly being dominated by machine learning techniques. In this paper we use and measure the effectiveness of three machine learning methods: a baseline linear learner of Support Vector Machines, a Fully Connected Neural Network, and a Convolutional Neural Network. Tested in Kaggle, we have achieved a 41%, 22%, and 92% respectively on a modified MNIST data set.**

I. Introduction

Machine Learning (ML) is quickly pushing the boundaries what is considered the state of the art in fields once dominated by classical techniques such as Computer Vision (CV) and Natural Language Processing (NLP). Machine Learning does so by taking advantage of advances in computing performance to perform rapid and complex pattern finding in data sets, which allow for day-to-day improvements in areas such as spam filtering, online search results, virtual assistants, and much more. One of the more common research directions in ML is CV, which involve finding patterns in images. We use three ML techniques to perform number detection on a modified MNIST set provided by the Computer Science department at McGill University. Our contributions are: a linear Support Vector Classification written in python using scikit-learn library, a hand written Neural Network (NN) written in python, and a Convolutional Neural Network (CNN) written in python using a PyTorch framework.

The Modified MNIST data set is a 64×64 grey-scale image provided as a .csv file. Each image has a background, and various MNIST numbers which are randomly rotated, scaled, and inserted onto the image. An example image can be found in Figure 1. The training data set has 50K images with labels and the test set has 10K images.

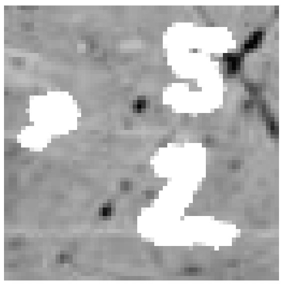


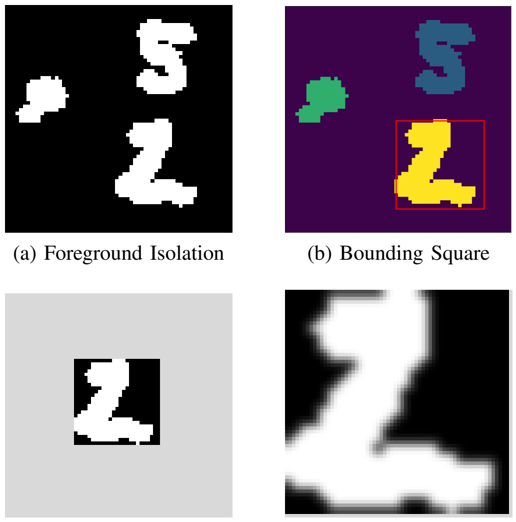
Fig. 1: Modified MNIST Image

II. Feature Design

In addition to the raw data set, the group also produced several modified data sets for either easier computation or higher accuracy through pre-processing feature extraction. The first modification was to the size of the data set. The original data set used floats for each pixel, but each value was only an integer. Re-saving this data set in the different format yielded about 65% reduction in total size.

The next step was a series of transformations onto the original data set which could help improve performance. For example, as can be seen in Figure 1, the background is grey-scale, while the numbers itself are pure white (RGB value of 255). Using a basic threshold filter, the background filter could be removed.

The largest number could be prematurely defined using a bounding square method, which isolated images based on the relative (x, y) area, as shown in Figure 2. The number was then visually isolated, and expanded until it was the size of the original image, after which it was fed through to the classifier. Note that the color for bounding box was added for clarity. These transformations were made available to for all the different types of classification.



(a) Foreground Isolation (b) Bounding Square

(c) Mask Square (d) Zoomed

Fig. 2: Pre-processing Techniques

III. Algorithms

A. A Linear Support Vector Classification

Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

The intuition behind SVMs can be visualize using the labeled training data below:

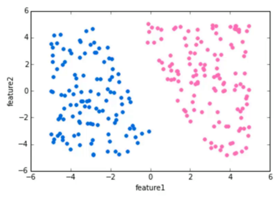


Fig. 3: Labeled Training Data

Intuitively we can draw a separating "hyperplane" between the classes.

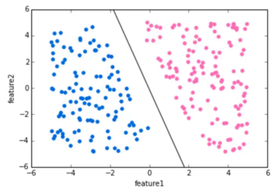


Fig. 4: Separating Hyperplane Line

However we also have many other options of hyperplanes that separate the classes.

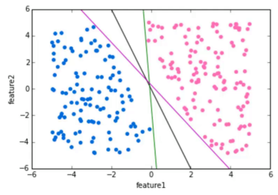


Fig. 5: Other Separating Hyperplane Lines

So the question rises. In order to determine the best line, we would need to choose a hyperplane that maximizes the margin between classes, in the Figure. 6 below it is denoted as the dotted line and a margin extends out from such hyperplane.

The vector points that the margin lines touch are known as Support Vectors in the Figure. 7.

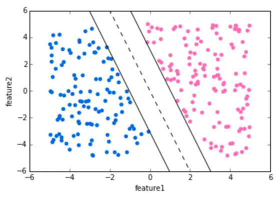


Fig. 6: Choose The Maximize Hyperplane

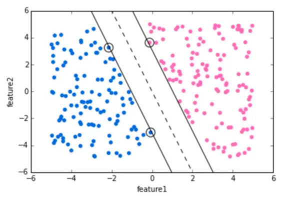


Fig. 7: Support Vectors

B. *Fully Connected Feed Forward Neural Network*

Feed Forward Neural Networks are a form of artificial neural networks with the property that no cycles exist between any of the layers of the network. Hence information moves in a single forward direction. Ironically, in our case information also flows backward due to the implementation of the back propagation learning algorithm applied.

The back propagation algorithm is used to determine the required gradient to be applied on the network weights. It involves making a forward pass through the network to the output softmax activated layer where a prediction is made. The softmax activation function gives class liklihood predictions. The negative log likelihood (multi-class cross entropy loss) is used to evaluate an error between the target labels and the predicted labels. At this point we backpropogate, in equation 1 we determine the gradient of the loss function with respect to the preactivation of the output.

$$E = -\log \sigma(t) = -\log(softmax(t)) \tag{1}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial s}{\partial t} = \frac{\partial}{\partial z_j} - \log(\sigma(t)) = ... = \sigma(j) - \delta_{jt} \tag{2}$$

Our algorithm in particular makes use of an elegant result capitalizing on the use of one - hot encoding of the target labels. In particular if we take the result from equation (2) and instead of considering each output neuron as j, we take t_j in vector notation as a one-hot encoded vector such that,

$$\frac{\partial E}{\partial z_j} = \sigma(j) - t_j \tag{3}$$

Then we are directly calculating the gradient of the loss function with respect to the preactivation of all output neurons. We are then able to update the corresponding weights, and repeat these steps on multiple epochs.

C. Convolutional Neural Network

CNNs are a subset within Neural Networks which use a convolution as the core operator. This allows the node to be able to determine spatial relationship in data. Aside from image processing and CV, CNNs have also been shown to provide excellent results in identifying patterns in time series data. CNNs are more or less the state of the art in performing image detection.

IV. Methodology

A. A Linear Support Vector Classification

Internal testing was done on a model initially, named internal SVC, and each hyperparameters were hand tuned until a result was found. The results were done by taking the training set and performing a $80/20$ split training/validation set segmentation. The sci-kit learn LinearSVC library was then implemented to fit the model.

The score on the initial model was poor, therefore the data sets were transformed and processed using our feature design written in Python, and a new model was created, named kaggle SVC, with the transformed data sets. LinearSVC library was used once again to fit this new model and the improvement in accuracy was astonishing.

While fitting the models with sci-kit learn SVM module, we were wondering what are the best parameters to use, so we used a sci-kit learn model selection method called grid search to find the right parameters (like what C values to use) by creating a 'grid' of parameters and trying out all the possible combinations and see what works the best, the grid search improved our accuracy by a little.

B. Fully Connected Feed Forward Neural Network

In order to determine hyperparameters for the feedforward neural network, cross validation against an $80/20$ split training / validation set was performed. Each hyper parameter was tuned separately with all other parameters kept constant.

TABLE I: Hyper-parameter Tuning

Hyper Parameter	Tested Range	Tested Increment	Percentage Increase (%)	Selected Value
lr	0.1 - 0.0001	*log*	+8.7	0.0001
l2 regu.	200 - 0.01	*log*	+10	100
hidd. nodes	10 - 70	1	+3.9	43

lr refers to learning rate, l2 regu. refers to l2 regularization and hidd. nodes refers to the number of nodes in the hidden layer. Rather than performing an exhaustive range search, a log range is used to explore a wider range of values.

C. Convolutional Neural Network

Internal testing was done purely by taking an initial model and hand tuning hyper-parameters until an optimal result was found. The results were done by taking the training set and performing a random $80:20$ training/validation set segmentation. The test set was never opened until final Kaggle submission. The validation of each variable was performed with 10 epochs, which looked at average accuracy on the validation set. The initial CNN architecture can be seen in Figure II. This was based on a small scale architecture which would preform well in MNIST, which under pre-processing the data set would then approximate.

TABLE II: CNN architecture

Stage	Type	Shape
1	Conv2d	(1, 10, 5)
2	MaxPool2d	(2, 2)
3	Conv2d	(10, 10, 5)
4	Linear	(1690, 50)
5	Linear	(50, 10)

The convolution shape is determined as the following tuple of (**input channel size, output channel size, kernel size**).

The following tests were performed with the best validation accuracy used to determine whether the feature was accepted. Each test was run multiple times to account for the variance in testing - typically a few percentage points difference meant that the added feature was not essential to improving performance. In these scenarios, the network was not modified. Variable tuning can be split into three sections: architecture modification, hyper-parameter tuning, and pre-processing. As seen in Table V, pre-processing modifications was the largest contributor to the improvement in accuracy. Since the initial architecture was geared towards MNIST, the amount of architecture design search was limited and a majority of the searches yielded low improvements at best. The benchmark default accuracy for an unmodified data set with the initial architecture is about 59%, which varies by up to 5%.

TABLE III: Architecture Modification

Parameter	Original Value	Tested Value	Percentage Increase (%)	Accepted
Conv1 Kernel	5	3	-5	No
FC1 Output Channel	50	100	+4	Yes
FC1 Output Channel	100	200	+2	No
Conv1 Output Channel	10	20	+4	No

TABLE IV: Hyper-parameter Tuning

Hyper Parameter	Tested Range	Tested Increment	Percentage Increase (%)	Selected Value
α	0.001 - 0.004	0.001	+4	0.001
α	0.0001 - 0.001	0.0001	+5	0.001
α	0.001 - 0.002	0.0001	+3	0.0012

Note that in Table V, the function zoom was not implemented without a specific target. Hence it typically follows after the number has been identified. Based on the tables, implementing binarization, masking and zooming in on the target image yields the greatest accuracy improvement, from 60% to over 90%. This matches well with the test results that were obtained by Kaggle.

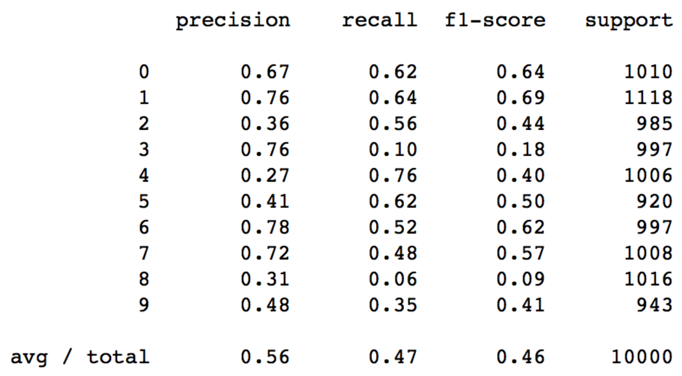
TABLE V: Pre-processing

Method #	Method Name	Percentage Increase (%)
1	binarize	+10
2	mask square	+20
3	Gaussian blur	-3
4	zoom	N/A
1, 2		+25
1, 2, 3		+30
1, 2, 4		+20

V. Results

A. Linear Support Vector Classification

Predictions were made after the training on the models, 41% accuracy was achieved on Kaggle. We used sci-kit learn metrics module to print out the confusion matrix which describe the performance of the SVM classification model on the test data for which the true values are known, and also generated a classification report to visualize our result.



```
             precision    recall  f1-score   support

          0       0.67      0.62      0.64      1010
          1       0.76      0.64      0.69      1118
          2       0.36      0.56      0.44       985
          3       0.76      0.10      0.18       997
          4       0.27      0.76      0.40      1006
          5       0.41      0.62      0.50       920
          6       0.78      0.52      0.62       997
          7       0.72      0.48      0.57      1008
          8       0.31      0.06      0.09      1016
          9       0.48      0.35      0.41       943

avg / total       0.56      0.47      0.46     10000
```

Fig. 8: SVM report

B. Fully Connected Feed Forward Neural Network

Using the *test_x.csv* dataset provided, predictions where made using the final hyperparameters tuned as shown in the preceding section. 22% accuracy was achieved after 1000 epochs of training as shown in TABLE VI below.

TABLE VI: NN Final Results

Epoch	Average loss	Accuracy (%)
1	90.22	9.85
50	33.84	11.95
100	13.06	14.46
150	5.45	16.88
200	2.68	18.34
250	1.67	17.91
300	1.32	18.61
350	1.18	19.41
400	1.14	19.52
450	1.14	19.99
500	1.13	20.44
1000	1.10	21.22

C. Convolutional Neural Network

Applying all of the optimal values found using architecture experimentation, hyper-parameter tuning, and pre-processing results in a validation set accuracy of approximately 92%. The final results can be seen in Table VII.

TABLE VII: CNN Final Results

Epoch	Average loss	Accuracy (%)
1	0.5191	86
2	0.3969	90
3	0.3709	90
4	0.3485	91
5	0.3278	92
6	0.3253	92
7	0.3169	92
8	0.3216	92
9	0.3159	92

D. Comparison among the different methods

We were interested on how other classifier would perform, so we also implemented the random classifier and majority-class classifier. And the following is what we have discovered:

TABLE VIII: Kaggle Scores Comparison

classifier	Accuracy (%)
Random	9.766
Major	9.266
internal SVC	10.7
kaggle SVC	41
internal SVC gridsearch	9.1
kaggle SVC gridsearch	41.8
NN	22
CNN	92

VI. Discussion

One area that could be improved would be the pre-processing. As noted in the pre-processing, the bounding squares were done using an (x, y) coordinate system. This works when the numbers are vertically aligned but result in errors when the numbers are tilted. For example, a 1 might be scaled larger than a corresponding 2, but because it is tilted at an angle, it's bounding box might smaller than the other, resulting in incorrect classification. Not because of the classifier, but the pre-processing stage might have incorrectly identified the wrong number to mask and to zoom in on. The method to improve this would be to use a bounding circle.

Whereas a bounding box is limited by the angle with which it is calculated, a bounding circle is rotation-invariant; that is, it will calculate the largest scaling regardless of the orientation of the number. While this was attempted, the only package which supported this operation was OpenCV, which did not have the required interoperability with Numpy's ndarray and pyTorch's Tensor object for this to be easily feasible.

Another approach would be to use the greatest distance between any two points in a number. For example, the number one has the greatest distance between the top and the bottom, whereas a 5 has the greatest distance along the diagonal.

Regardless of it's rotation, determining this greatest distance would allow the mask to select the number that has been scaled up the most.

Other improvements are with regard to the hyper-parameter and variable optimization. Different CNN architectures have been proposed with features such as pointwise convolution, with varying feature size and pooling techniques. Without a longer period of time, these variables are unable to be iterated and implemented upon. In addition, there was no exhaustive search over the design space to indicate whether the hyper-parameters were at a local or a global minima. Indeed, by iterating with smaller step sizes, it is possible that the tradeoff between exploration and exploitation was not managed correctly, and as a result some options were missed.

VII. Statement of Contributions

Each member of the group was assigned with different methods to work on, we all met weekly and discussed on how to approach the problem and contributed in developing the methodology. Everyone coded their solutions individually and we used a private GitHub repository to store information and data inputs/outputs. The report was written under the collaboration of three of the group members and we hereby state that all the work presented in this report is that of the authors.