COMP 551-001 Kaggle Competition: Classification of Modified MNIST*

Yingnan Zhao[1], Vincent d'Orsonnens[2] and Hamed Layeghi[3]

***Abstract*— This paper provides the report for the Kaggle Competition (assignment 4) of COMP 551 using the provided Modified MNIST dataset. The dataset includes a set of 8-bit grayscale images that include 2 or 3 digits of different sizes that are rotated and scaled from the classic MNIST dataset. The goal is to design Machine Learning algorithms that identify the biggest digit in each image. Several algorithms have been used in the report. First, a linear SVM is used which lead to relatively low accuracy. Second, a fully connected neural network completely developed by the team was implemented. Finally, a convoluted neural network was trained and tested on the preprocessed dataset which showed the best performance.**

I. INTRODUCTION

The MNIST database [1] is a set of handwritten images that is popular for training and testing of Machine Learning algorithms [2]. The provided images in the Modified MNIST include 3 numbers that are rotated and scaled from the MNIST dataset and are written on random backgrounds. Some samples of the train dataset with their associated outputs are shown in Figure 1.

The format for the images is 8-bit grayscale image, thus each pixel has 256 shades of gray represented by numbers 0 (black) to 255 (white) as shown in Figure 2.

50000 training examples and 10000 test examples are provided. The labels to the training set is known but the labels for the test set is not available.

In this report, three different classifiers are implemented to recognize the biggest digit in each image. First, a linear classifier, namely linear SVM, is tested which gives near 50% accuracy which shows the data is not linearly separable. Next, a feedforward neural network classifier is tested which leads to acceptable performance. The best performance is achieved by convolutional neural network with 10 layers.

The remaining part of the report is organized as follows. Section II explains the preprocessing methods used on the original dataset. Section III reviews each of the three classifiers used. Next, in section IV, we explain how the different hyperparameters are selected for each classification algorithm. In section V, the resulting accuracy and confusion matrices for each method are reported. Finally, section VI discusses the results obtained in the previous section.

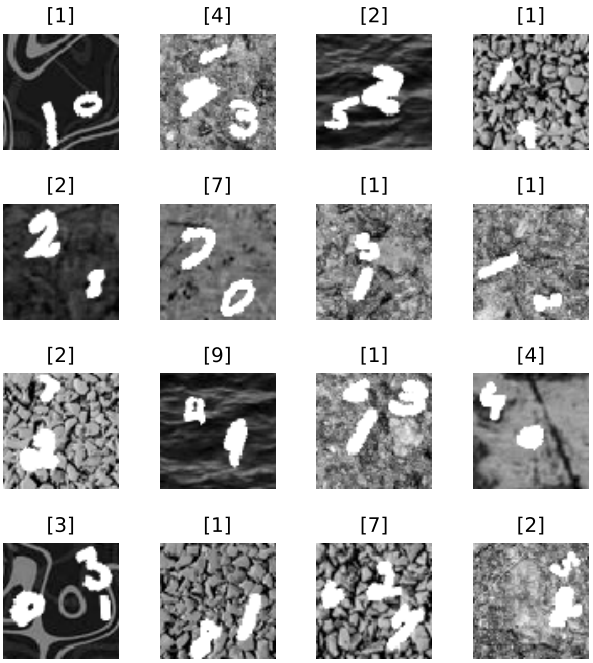

Fig. 1. 25 Random Samples of the original train dataset

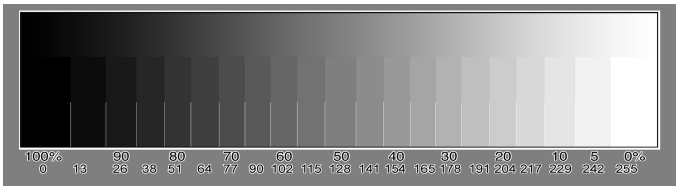

Fig. 2. 8-bit Grayscale Shades of Gray

II. PREPROCESSING

Before the data is used for training, it is better to use some preprocessing techniques to make the learning process easier and to increase the accuracy. To this end, the following image processing filters were used.

A. Thresholding

Since the numbers in the dataset match the 255 shade, a simple idea for preprocessing is to use *image thresholding*. The idea of thresholding is to compare the pixel values of the input image f with some threshold T and make a binary

*Kaggle Team name: BetaGo, Best Score: 0.96928

[1]Yingnan Zhao, Student ID: 260563769, Electrical Engineering, McGill University, nan.zhao2@mail.mcgill.CA

[2]Vincent d'Orsonnens, Student ID: 260746099, Software Engineering, McGill University, vincent.dorsonnens@mail.mcgill.ca

[3]Hamed Layeghi, Student ID: 260524499, Electrical Engineering, McGill University, hamed.layeghi@mail.mcgill.ca

decision for the output binary image g as below

$$g(i,j) = \begin{cases} 1 & f(i,j) \geq T \\ 0 & f(i,j) < T \end{cases} \quad (1)$$

for all i, j where i, j represent the coordinates of the ij^{th} pixel [3].

The output of this filter is shown in Figure 3

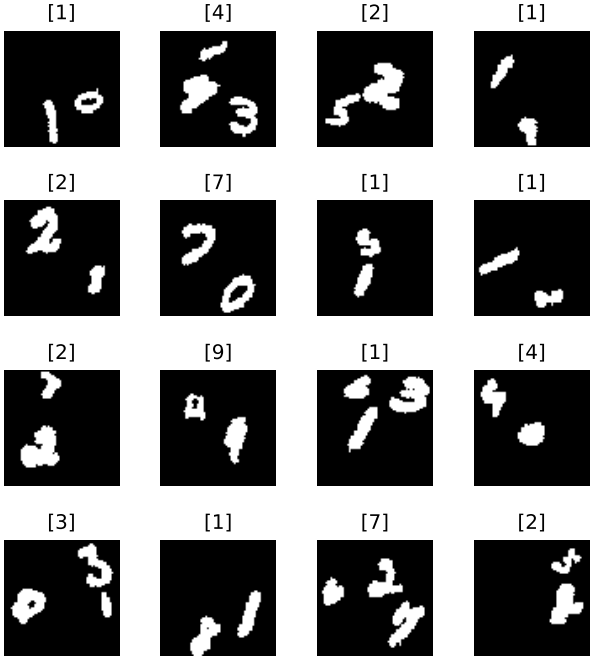

Fig. 3. Output of thresholding on images with $T = 255$ from Figure 1

B. Median Filter

As can be seen in Figure 3, there are some small white areas in some of the images that can act as undesirable noise. Median filtering is one method to remove the noise from the images and smoothen the edges. The main disadvantage of median filter is the damaging of thin lines and sharp corners [3].

The idea of median filter is to choose the median of the pixel values in a neighborhood of the given coordinate. These neighborhoods could be defined as disk, square, or any other shape of interest.

The output of the median filter using a disk of radius 1 applied on thresholded images are shown in Figure 4.

C. Biggest Digit

As the labels represent only the biggest digit in each digit, one way to filter the unnecessary information from each image is to find the biggest digit in each image using existing libraries. We used `skimage` library to extract the biggest digit in each image. The following algorithm was used for this purpose.

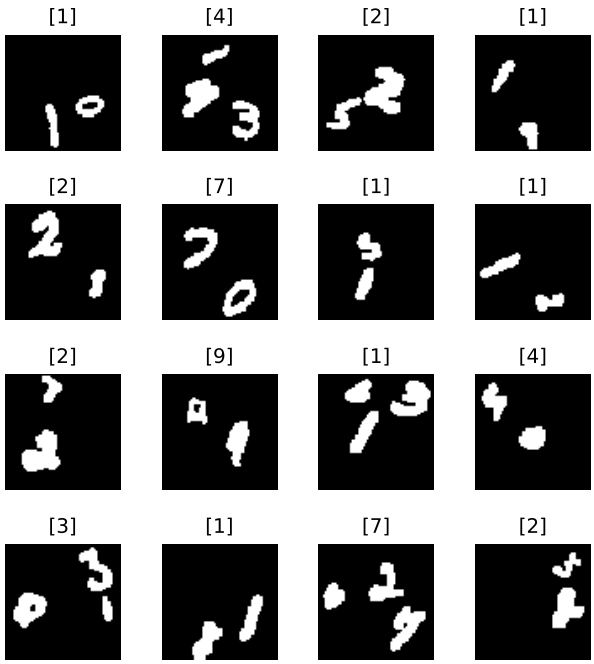

Fig. 4. Output of median filter on thresholded images from Figure 3

Algorithm 2.1: Given the image I, use the following steps to find the biggest number

Step 1 Apply the threshold filter to obtain I_T.

Step 2 Apply the median filter with disk radius $r = 1$ to obtain I_M.

Step 3 Use `skimage.morphology.closing` on I_m to find the rectangles that close each separate segment (digit).

Step 4 For the rectangle i with length l_i and width w_i, $l_i \geq w_i$, find $a_i = l_i^2$ and find the maximum $a_{max} := \max_i a_i$ and the associated l_{max} and w_{max}.

Step 5
- (i) If $l_{max} <= 28$, find the maximum area and cut the image to that rectangle.
- (ii) If $l_{max} >= 29$, the digits have merged, use `skimage.morphology.erosion` with `disk(1)` on I_T and go to Step 3. If this happens again, ignore and go to the next Step.

Step 6 Expand (or cut) the image to 32×32 image size.

The 28 pixel criterion above has been found by checking several random images that haven't been merged and it could be evaluated by the 120% maximum scaling which has been done on the original MNIST dataset.

The output of this filter on thresholded images are shown in Figure 5. As can be seen, the filter works very well for the The main advantage of this filter apart from removing the misleading small digits for the learning algorithm is that it reduces the size of the features to one forth of the original data. The main disadvantage is when the digits are on top of each other or can not be separated easily, we might have written forms which do not represent real numbers.

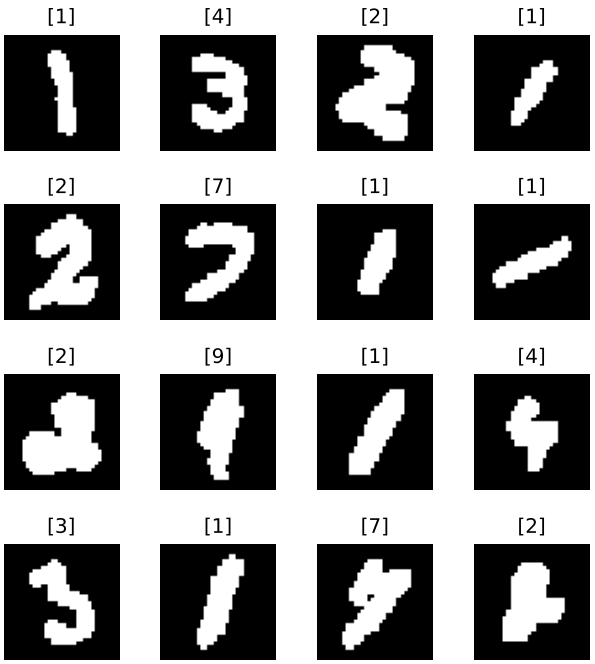

Fig. 5. Output of biggest digit filter on thresholded images from Figure 3

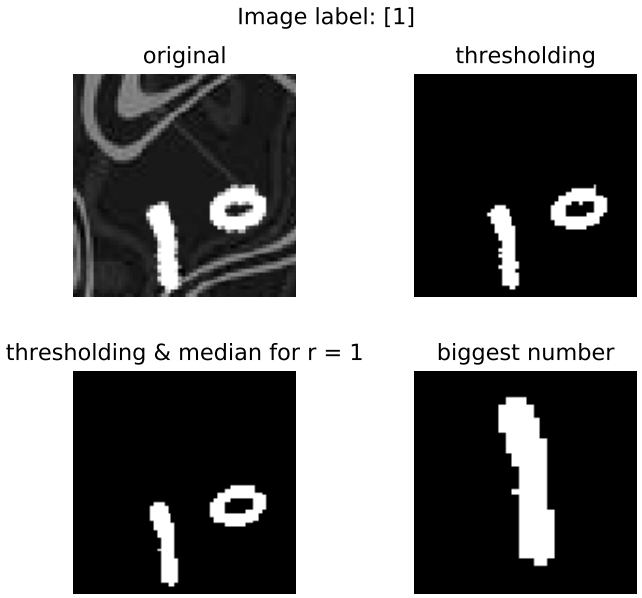

Fig. 6. Output of Different Filters Applied on the first image in Figure 1

D. Applied Filters

Figure 6 shows the effect of all the above filters on one image. It can be seen that the threshold and median filter with disk radius $r = 1$, and also the biggest digit filter work the best visually. In application, however, it was found out that the thresholded data with $T = 230$ works better for CNN since it adds some noise to the data that is not necessarily bad for training as it could help the model skip the local minima.

However, for Linear Classifier and Feedforward Neural Network (FFNN) as the best performance was not the goal, the biggest digit filter which needed less features and training time was applied.

III. METHODOLOGY

To start with, we had to split the original dataset into a training and validation set. The test set was provided without labels, we had to submit our predictions to Kaggle to get the accuracy of our models. We decided to go with a simple 95/5 split for the training and validation sets. Then we applied our preprocessing steps defined in the Feature Design section to generate 2 new datasets based on the original one.

- For the first new dataset, we applied the threshold filter.
- For the second dataset, we applied the threshold and median filters, then extracted the biggest number.

So we had three datasets now, each consisting of 47,500 training examples and 2,500 validation examples. We could then compare which of the preprocessing and feature selection steps work best for each model.

A. Linear SVM

Support Vector Machine (SVM) classifiers became popular in the last few decades. They are grouped as sparse kernel machines and have the advantage that their parameters can be found by a convex optimization problem and so any local solution is also a global optimum [4]. In linear Support SVM, we use a classifier of the form

$$y = \mathbf{w}^T \mathbf{x} + b \tag{2}$$

The goal is to minimize the following loss function

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2} \|\mathbf{w}\|^2 \tag{3}$$

where ξ_n are the slack variables that define the class of each point shown in Figure 7. $C > 0$ controls the trade-off between the slack variable penalty and the margin. Since $\xi_n > 1$ for any point that is misclassified, it follows that $\sum_{n=1}^{N} \xi_n$ is an upper bound for the number of misclassified points. As $C \to \infty$, the classification is only possible for linearly separable data and it represents the case where no misclassified point is tolerated.

B. Fully Connected Neural Network

The second algorithm we used to try to solve this classification problem is a fully connected neural network implemented from scratch using the Numpy library. A feed-forward neural network works by passing information from one layer to the next until the last layer where it makes a prediction, then it uses the error on that prediction to update its weights in order to reduce the error. The information in the network propagates forward as follow: $a_{l+1} = \sigma(W^{(l)T} a_l)$

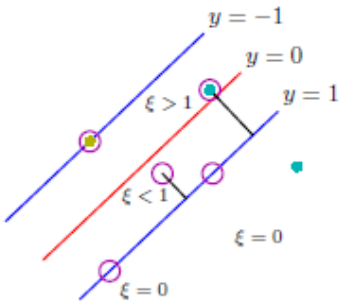

Fig. 7. Illustration of the slack variables $\xi_n \geq 0$ [4, p. 332]

where a_l is the activation of layer l, σ is a non-linear element-wise function and $W^{(l)}$ represents the weights between each neuron of layer l and $l+1$ [5].
Since we are dealing with a 10-class classification problem, the last layer of our network is a 10 neurons soft-max layer which outputs a probability distribution, the neuron in the final layer with the highest activation value can then be interpreted as the class with the highest probability. It then makes sense to use the cross-entropy loss function defined as

$$loss = -\sum_{k=1}^{K} Y_k \log P_k$$

where Y is the target and P the prediction. For all the networks we tried, we used the $tanh$ activation function for every hidden layers and trained them on the second dataset described earlier, with the biggest number extracted, because the number of features is reduced and it's less expensive to train. We used gradient descent in conjunction with back-propagation to train the networks.

C. Convolutional Neural Networks

A convolutional Neural Network (CNN) is a class of deep feed-forward neural networks that has great success in the field of analyzing visual imagery [6]. It shares certain similarities with a regular neural network, it is made up of neurons that have learnable weights and biases. However, it arranges its neurons in a 3-dimensional space and reduce the width and depth of the input image so that less neurons are needed. It is mainly consist of three types of layers, the convolutional layer, pooling layer and fully connect layer, the network can be built by stacking those layers sequentially [7].

When it comes to image recognition, a convolutional neural network is one of the best options. For this project, a 10-convolutional layer CNN was built. There are 5 max pooling layers in total, one after every two convolutional layers to reduce the spatial size, each pooling layer will exactly reduce the input length and width by half. Also, there are 5 dropout regulations being applied, one after every two convolutional layers which has a dropout probability of 0.25, and one dropout after the fully connected layer which has a drop out probability of 0.5, the team believe this will prevent overfitting and reduce the interdependent learning amongst the neurons [8]. As shown in the Figure 8, there are max pooling layers after convolutional layer 2, 4, 6, 8, 10, and dropout regulations after convolutional layer 3,5,7,9 and the fully connected layer. Activation function Rectified Linear Unit (ReLU) is used after every convolutional layer and the fully connected layer, and batch normalization is applied after the activation function. Because a CNN is able to learn filters on its own, we decided to use the simple thresholded dataset to train it. The CNN has a huge amount of parameters and the team wants to use as many data as possible to train the model. Moreover, 5% of the original training dataset is still 2500 data points, it can produce a reasonable result in a short amount of time. Data is fed into the CNN network a batch at a time, using the pytorch data loader class, and the batch size is set to be 300.

IV. HYPERPARAMETER SELECTION

A. Linear SVM

Because of the computational difficulty of training, only 5 values of C on a logarithmic scale from 0.001 to 1 were chosen for tuning.

B. Fully Connected Neural Network

We decided to keep the architecture of our neural network simple, since it is computationally expensive to train. So we only used 1 hidden layer and we decided try 4 different values for the number of neurons in the hidden layer, (100, 200, 300, 400). To train the networks, we tried two different values for the learning rate, 0.1 and 0.03 and picked the combination giving the best result based on the validation performance.

C. Convolutional Neural Network

As shown in the Tables I and II there are close to 100 hyper-parameters, it is impossible to tune them all. The first filter size is chosen to be 5 as recommended by [9], on MNIST dataset 5x5 filter size gives the best result as shown in Figure 10. The study also showed that there is a trade of between complexity and accuracy, but if the model gets too complex without sufficient data points it has the risk of over-fitting. The number of convolution layers was chosen by gradually increasing the number of convolution layers and record the performance on the validation set. It was found through several trainings with different models that 10 convolution layers yields the best result. This can be seen in Figure 11 where the performance increases until it reaches 12 layers, where it overfits.

A pooling layer was used to decrease the number of neurons towards the bottom as suggested by [10]. Specifically, after every two convolution layers a pooling layer will be employed to reduce the number of neurons by half (decrease

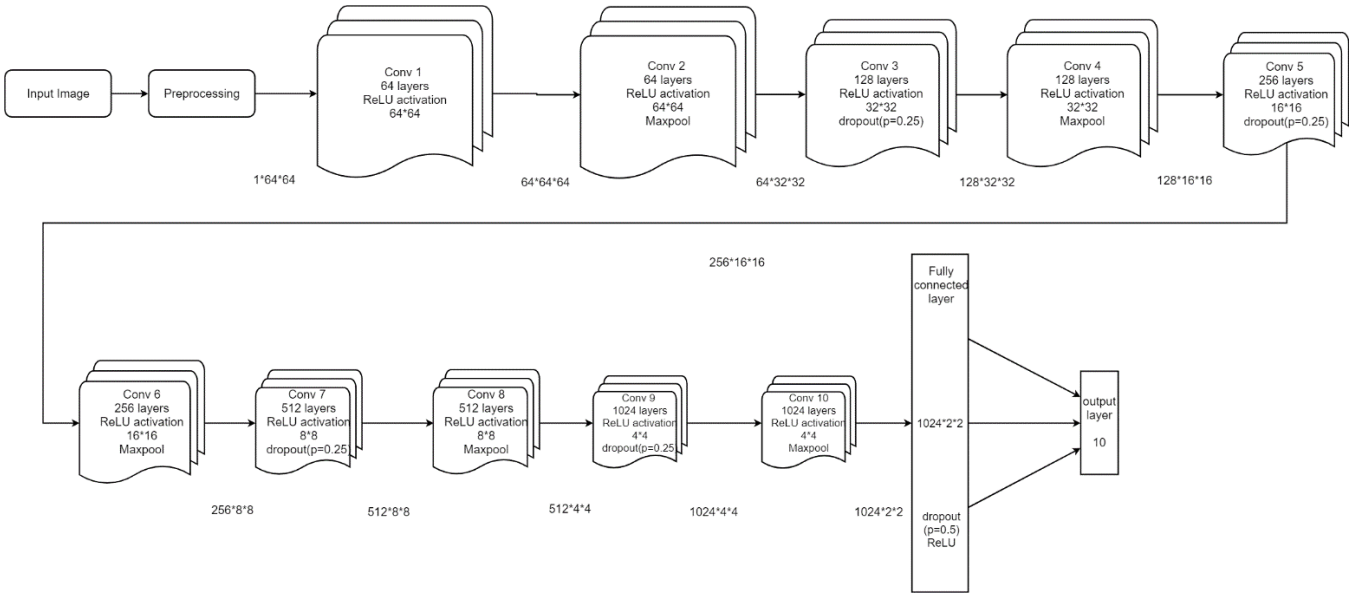

Fig. 8. Architecture of CNN model 8

TABLE I
HYPER-PARAMETERS FOR CNN

	cov 1	cov2	cov 3	cov 4	cov 5	cov 6	cov 7	cov 8	cov 9	cov 10	fully connected
number of filters	64	64	128	128	256	256	512	512	1024	1024	n/a
filter size	5	3	3	3	3	3	3	3	3	3	n/a
stride	1	1	1	1	1	1	1	1	1	1	n/a
zero padding	2	1	1	1	1	1	1	1	1	1	n/a
max pooling	no	yes	no	yes	no	yes	no	yes	no	yes	n/a
kernel size	n/a	2	n/a	2	n/a	2	n/a	2	n/a	2	n/a
stride	n/a	2	n/a	2	n/a	2	n/a	2	n/a	2	n/a
dropout	no	no	0.25	no	0.25	no	0.25	no	0.25	no	0.5
activation function	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU

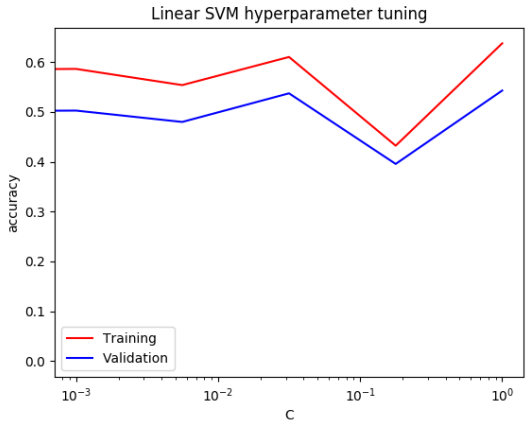

Fig. 9. Training and validation performance for different value of C

both length and width by half and increase the depth by 2). For the number of filters for the first convolution layer (the number for the rest of the convolution will just double every two layers), different values were tested, the accuracy was improving when the number of filters changed from 8 to 64 for the first convolution layer, and the improvement was not very significant by increasing it over 64, but the computing time increased exponentially as the number of the filters were increased.

Dropout is proven to be an efficient and effective strategy to prevent overfitting [11], thus it is used as the regulation strategy. For this purposes, 25% of tensors get dropped every 2 convolution layers. Adam optimization method was used because it has an adaptive learning rate and momentum since it is shown to be more effective than other methods [12].

V. RESULTS

A. Linear SVM

The accuracy on training and validation sets for different values of C are shown in Figure 9. The best accuracy was achieved for $C = 1$. The resulting SVM classifier achieved an accuracy of 0.543 on the validation set. It could be seen that the accuracy of the linear classifier does not vary a lot with the parameter tuning and that is because the linearity assumption limits the performance of the classifier to a certain maximum. Figure 12 shows the confusion matrix for this model.

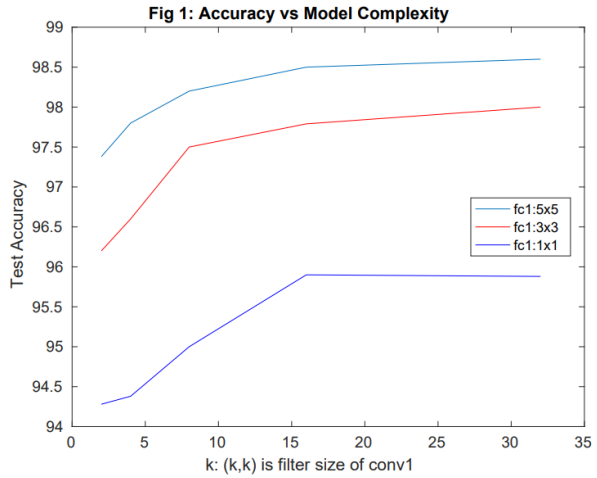

Fig. 10. Accuracy of CNN vs Model Complexity [9, p. 4]

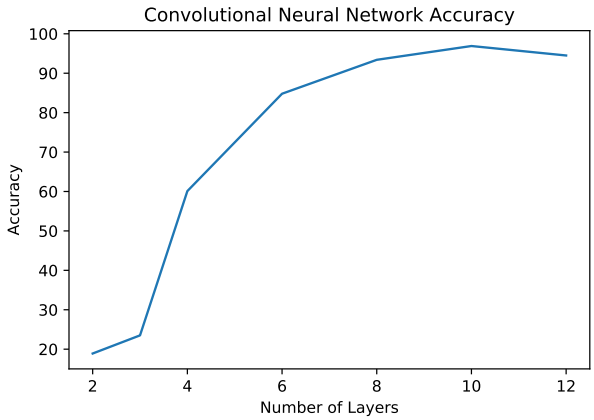

Fig. 11. Effect of Number of Layers on Convolutional Neural Network Accuracy

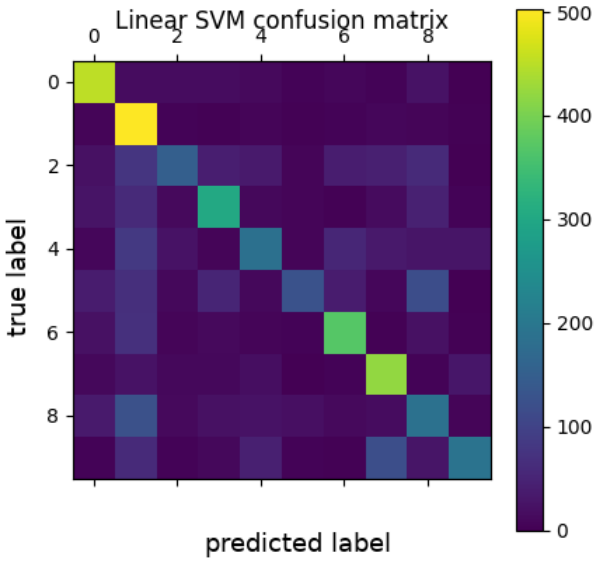

Fig. 12. Confusion matrix for the best SVM model (targets vs outputs) with 2500 validation examples

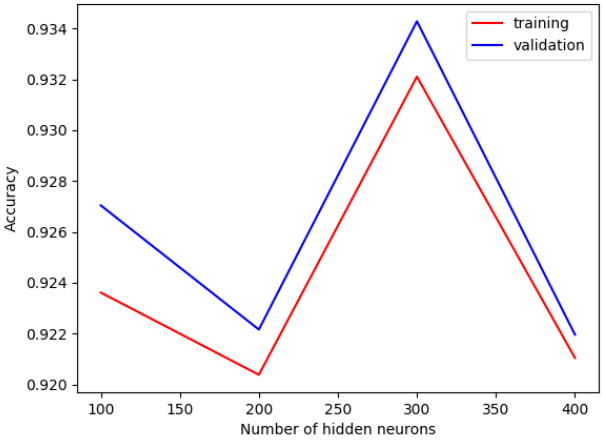

Fig. 13. Validation accuracy for different layer size

B. *Fully Connected Neural Network*

Using every combinations of learning rates and layer architectures to find the best model, we found that the best accuracy on the validation data was obtained by the network composed of 1 hidden layer with 300 neurons and a learning rate of 0.03. For all the network configurations that we tried, the learning rate of 0.1 was too high. The best accuracy achieved on the validation data using this network is 0.934. Figure 13 shows the accuracy for the different models we tried. Figure 14 show the confusion matrix of the best model. We see a very high accuracy for the digits 0 and 1, which are the easiest to classify and a low accuracy for the digits 2 and 5. Also, the network tends to misclassify an 8 by a 5 quite often.

C. *Convolutional Neural Network*

Using the 10-layer CNN model has both advantages and disadvantages. The CNN network can be very accurate on image recognition tasks, our model is able to achieve an accuracy of 96.93% training on only 47500 samples, while there was still room for improvement by adjusting the hyperparameters. The drawback of the model is its complexity, and the fact that it needs a lot of computational budget and memory to train. According to Dhingra [9], there is a trade off between complexity and accuracy. Since there is no requirement about the speed or complexity in the assignment handout, a complex model with 10 layers was chosen. The confusion matrix for this model is shown in Figure 15 for model 8 in Table II.

TABLE II

SPECIFICATION OF THE TRAINED CNN MODELS AND THEIR ACCURACY ON VALIDATION SET

model	layers	regulation	max pooling layer	preprocessing	accuracy
1	2	n/a	1	n/a	18.9
2	3	n/a	1	n/a	23.5
3	4	3 dropouts	2	n/a	60.1
4	6	3 dropouts	3	n/a	84.8
5	8	4 dropouts	4	threshold	93.4
6	10	5 dropouts	5	threshold	96
7	12	6 dropouts	6	threshold	94.5
8	10	7 dropouts	5	threshold	95.4

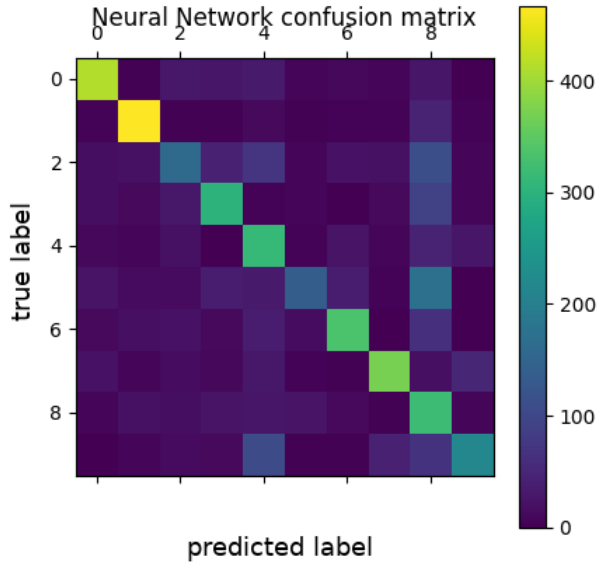

Fig. 14. Confusion matrix for the best FNN model (targets vs outputs)

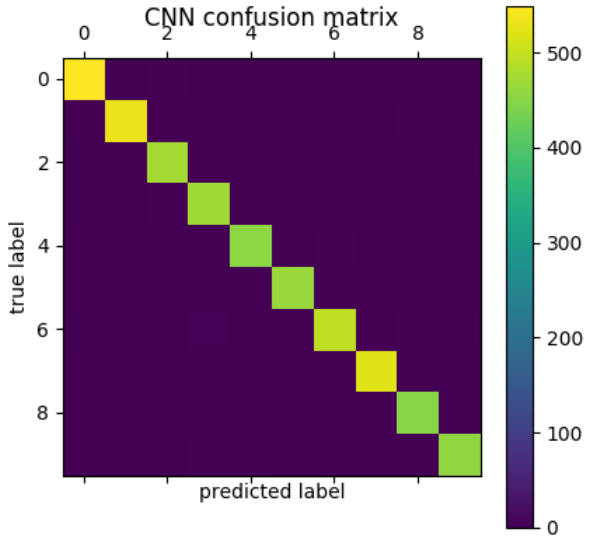

Fig. 15. Confusion matrix for the best CNN model

TABLE III

PERFORMANCE OF DIFFERENT CLASSIFIERS FOR MODIFIED MNIST

Classifier	Dataset	Parameters	Accuracy
Linear SVM	Biggest	C=1	0.593
Forward NN	Biggest	300 hidden neurons	0.934
CNN model 8	Biggest	see Table I&II	0.9693
CNN model 8	Threshold	see Table I&II	0.9693

VI. DISCUSSION

Table III gives a summary of all the classifiers discussed in the report

Choosing to use the biggest number dataset for the Linear SVM training and the Fully Connected Network training was a good choice because it's the dataset with the least number of features, which can help reduce overfitting, and it makes the training faster on our computationally restricted devices.

For the Linear SVM model, the accuracy score is the lowest of the three approaches. This was predictable since it assumes that the data is linearly separable, which is probably not the case for this problem. The confusion matrix in Figure 12 suggests that the digits 0, 1, 7, 6 and to some level 3 are able to be found by an acceptable accuracy using a linear classifier. However, the remaining digits 8, 5, 9, 2 and 4 are not predicted very well by linear SVM. The confusion matrix shows that 5, 8 and 9 are often misclassified by 8, 2, and 7, respectively which can be justified by their similar forms. Many numbers are misclassified to 1 and 8.

For the fully connected network, we see in Figure 14 a very high accuracy for the digits 0, 1 and 7, which are the easiest to classify because of their geometric form and a low accuracy for the digits 2 and 5. Also, the network tends to misclassify 9 by 5 and 5 by 8 quite often. In general, a lot of numbers are wrongly classified as 8 and then 5 which suggests that these are the two numbers which were not correctly learned. One improvement we could try is to add another hidden layer to our network.

For the CNN model, the confusion matrix in Figure 15 shows near perfect performance. The digits 0,1, and 7 are

easier to learn for CNN and yet the other digits show very good prediction as well.

STATEMENT OF CONTRIBUTIONS

Yignan implemented the convolutional neural network. Vincent implemented the fully-connected feedforward neural network. Hamed implemented preprocessing. Hamed and Vincent implemented the linear classifiers. Yignan and Vincent performed the heavy computational simulations on Amazon AWS. All three contributed to the report with Hamed being responsible for organizing and merging the final report.

REFERENCES

[1] Y. LeCun, C. Cortes, and C. J. Burges. Mnist handwritten digit database, yann lecun, corinna cortes and chris burges. [Accessed 16-March-2018]. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[2] W. contributors. (2018) Mnist database — wikipedia, the free encyclopedia. [Accessed 16-March-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MNIST_database&oldid=829836442

[3] A. C. Bovik, *The essential guide to image processing*. Academic Press, 2009.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[6] W. contributors. (2018) Convolutional neural network — wikipedia, the free encyclopedia. [Accessed 22-March-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=830433988

[7] A. Karpathy. Convolutional neural networks (cnns / convnets). [Accessed: 22- Mar- 2018]. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[8] A. Budhiraja, "Dropout in (deep) machine learning," 2016, [Accessed 22-March-2018]. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[9] A. Dhingra, "Model complexity-accuracy trade-off for a convolutional neural network," *arXiv preprint arXiv:1705.03338*, 2017.

[10] S. Lattner. [Accessed: 22- Mar- 2018]. [Online]. Available: https://www.researchgate.net/post/Could_you_give_me_some_advices_about_how_to_improve_the_performance_of_Convolutional_Neural_Networks

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[12] A. S. Walia. (2017) Types of optimization algorithms used in neural networks and ways to optimize gradient descent. [Accessed: 22- Mar- 2018]. [Online]. Available: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f.