COMP 551 Kaggle Competition: Classify hand-drawn pictures

Yutong Yan[1], Bridget Liu[2] and Yang Lu[3]

Abstract— **This paper provides the report for the Kaggle Competition of COMP 551 using the provided modified dataset as the subset of Google *Quick, Draw!* project dataset. The dataset includes a set of noisy images that include a hand-drawn object or noise only in total of 31 categories. The goal is to design Machine Learning algorithms that identify the hand-drawn object in each image. Three algorithms have been used in the report. First, a linear SVM is used. Second, a fully connected feed forward neural network implemented by hand is utilized. Finally, a convoluted neural network was trained and tested on the pre-processed dataset. Among the three, convolutional neural network performs much better than linear SVM and the feed forward neural network.**

I. INTRODUCTION

Quick, Draw! is a game built with machine learning. You draw, and a neural network tries to guess what youre drawing. Google has collected a huge dataset of hand-drawn images through the game. The provided noisy images in the modified dataset include a hand-drawn object or noise only. Some samples of the train dataset with their associated outputs are shown in Figure 1.

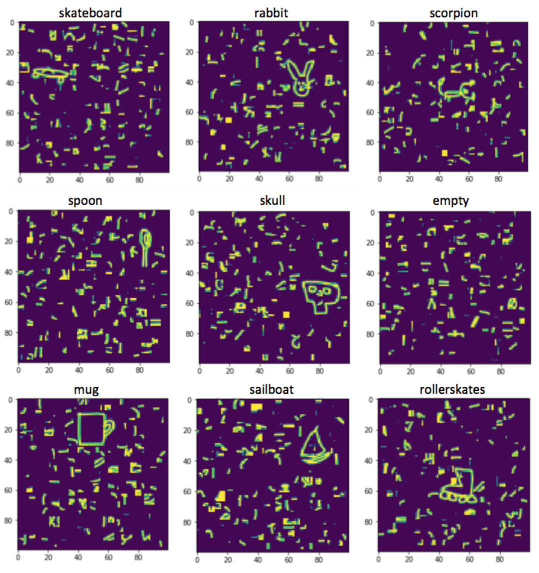

Fig. 1. 9 Random Samples of the original train dataset

*Kaggle Team name: 404

[1]Yutong Yan, Student ID: 260716973, Computer Science, McGill University, yutong.yan@mail.mcgill.ca

[2]Bridget Liu, Student ID: 260705926, Computer Science, McGill University, yitong.liu@mail.mcgill.ca

[3]Yang Lu, Student ID: 260631276, Math, McGill University, yang.lu4@mail.mcgill.ca

10000 training examples and 10000 test examples are provided. The labels to the training set are known but the labels for the test set are not available. Each image is 100px x 100px and stored as numpy array.

In this report, three different classifiers are implemented to recognize the hand-drawn object category in each image. First, a linear classifier, namely linear SVM, is tested which gives near 40% accuracy which shows the data is not linearly separable. Next, a feed forward neural network classifier is tested which leads to acceptable performance near 50%. The best performance was 80% and is achieved by convolutional neural network.

The remaining part of the report is organized as follows. Section II explains the preprocessing methods used on the original dataset. Section III gives an overview of the learning algorithms of the three classifiers used. Next, in section IV, we explain how the different methods are selected for each classification algorithm. In section V, the results for each method are reported. Finally, section VI discusses the results obtained in the previous section and future work.

II. PREPROCESSING

Since the original given dataset contains noisy images, it is better to preprocess the raw images to make classifier perform better to increase the accuracy. The following techniques have been 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 with some threshold and in the end make a binary decision for the output binary image, if the pixel value is larger than the threshold, the pixel value is replaced by 1, otherwise 0.

B. Extracting Largest Contours and Resizing

As the labels correspond to only objects excluding the noise, one way to filter the noise from each image is to use implemented function to find the biggest digit in each image using existing libraries. We used `opencv` library to extract the five biggest contours in each image. We only want to select the one with biggest area. This can increase the accuracy of the classifier dramatically. The following algorithm was used for this purpose.

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.

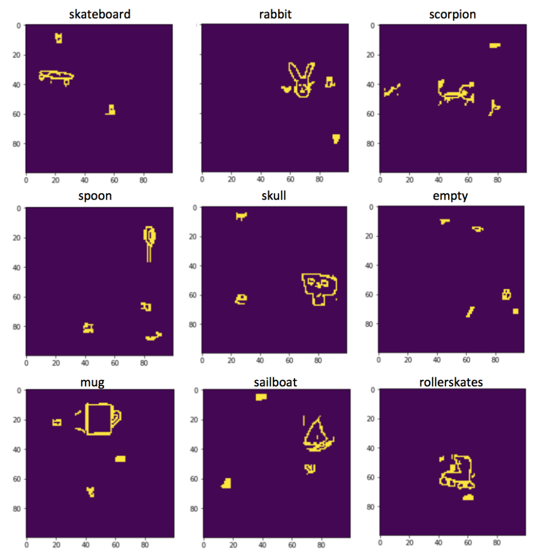

Fig. 2. Output of preprocessing on images

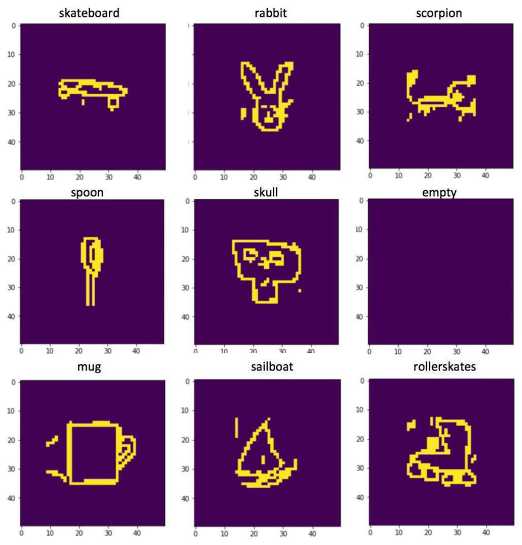

Fig. 3. Output of preprocessing and resizing on images from Figure

Step 2 Use `cv2.findContours` on I_m to find the five biggest contours.

Step 3 For each contour, the one with maximum area (width * length) is picked.

Step 4 If the width and length of all extracted contours are both less than 15 pixels, we choose to leave the image empty.

Step 5 If the image is not empty, we save the biggest contour, and we pad the zeros around the image to resize the image $50px \times 50px$.

The 15 pixel criterion above has been found by randomly checking raw images after image thresholding. By observation, the width and length of noise are less than 15 pixels. The output after preprocessing is shown in Figure 2.

The output after preprocessing and resizing is shown in Figure 3.

As can be seen, the preprocessing method works very well.

The main advantage of this Largest Contours apart from removing the misleading noise for the learning algorithm is that it reduces the size of the features to half of the original data.

The main disadvantage is that if the noise is connected with the contour we desire to extract, the noise will also be included in the extracted part.

III. Algorithms

A. Linear SVM

Support Vector Machine (SVM) classifiers became popular in machine learning field for a long time. 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. In linear Support SVM, we use a classifier of the form

$$y = \mathbf{w}^T\mathbf{x} + b \tag{1}$$

The goal is to minimize the following loss function

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

$C > 0$ controls the trade-off between the slack variable penalty and the margin.

B. Feedforward Neural Network

The second algorithm we used is a fully connected feed forward neural network, trained by backpropagation, implemented from scratch using the Numpy library. How a feed-forward neural network works is that it is initiated with random weights and it passes information from one layer to the next until the last layer where it makes a prediction, then it uses the error computed on that prediction to update its weights to reduce the error. The information in the network propagates forward as follow: $a_{l+1} = \sigma(W^{(l)T}a_l)$ 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 $l+1$.

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 computer vision. 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 reduces the width and depth of the input image so that less neurons are needed. It mainly consists of three types of layers, the convolutional layer, pooling layer and fully connected layer, the network can be built by stacking those layers sequentially.

IV. Methodology:

A. Train/Valid Split

To start with, we had to split the original dataset into a training and validation set. The test set comes with no labels, therefore to validate the results and observe how well the model is, we have to split the training and testing with a 75/25 split for SVM and CNN, a 80/20 split for feed forward neural network to better perform cross validation. Then we applied the preprocessing techniques to the raw images and generate the new dataset.

B. *Hyperparameter Selection*

1) Linear SVM: Ten values of C on a logarithmic scale from 0.00001 to 10000 were chosen for tuning.

2) Feedforward Neural Network: Three hyperparameters we used here are number of nodes, layers, and learning rate. For number of nodes, we used 100, 200, 300, and 400 for different values of neurons in one hidden layer. For learning rate, 0.001, 0.01, and 0.1 were chosen. We only used one hidden layer to tune number of nodes and learning rate to make it computationally efficiently. For layers, we used [2500, 400, 31], [2500, 400, 300, 31], and [2500, 400, 300, 200, 31] as one hidden layer, two hidden layers, and three hidden layers, since we acknowledged that 400 hidden nodes performs the best from the first experiment.

3) Convolutional Neural Network: There are much more hyperparameters that need to be considered for CNN model. We have considered the following hyperparameters:
Learning rate: four values of learning rate on a logarithmic scale from 0.0001 to 0.1 were chosen for tuning. It turns out 0.001 is the best since other learning rates will cause the model to either overfit on training data or converge too slowly.
Number of epochs: number of epochs is slightly different for each model with different values of layers, but generally from 10 to 30 is sufficient.
Batch size: values of batch size, 32, 64, and 128, have been considered. 32 performs better than the other two.
Activation function: activation functions sigmoid, tanh, and 'ReLU' have been considered. To save computational time, we decided to use 'ReLU' only.
Dropout rate: dropout is proven to be an efficient and effective strategy to prevent overfitting. Values of dropout rate 0.25 and 0.5 have been considered. Dropout rate of 0.25 for the first few layers and 0.5 for the last has been observed to perform the best.
Convolution layers: the number of convolution layers was chosen by gradually increasing the number of convolution layers and record the performance on the validation set.
Pooling layer: max pooling layer was used to decrease the number of neurons towards the bottom. Specifically, after every two convolution layers a pooling layer will be employed to reduce the number of neurons by half.
Filters: for the number of filters for the first convolution layer, same values were tested, the accuracy was improving when the number of filters changed from 15 to 3 for the first convolution layer, but the computing time increased exponentially as the number of the filters were increased.
Optimization method: Adam was used because it has an adaptive learning rate and momentum since it is shown to be more effective than other methods.

As shown in the Figure 8, there are max pooling layers after every two convolutional layers and dropout regulations after maxpooling layers and second last dense layer 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.

C. *Regularization Strategy*

1) Linear SVM: Due to the purpose of the linear SVM, regularization was not used on our baseline linear learning model.

2) Feedforward Neural Network: In order to avoid over-fitting, instead of directly using regularization techniques, we observed the training and validation accuracy curves during the training process.

3) Convolutional Neural Network: Dataset Augmentation: Data augmentation is a way we can reduce over-fitting the training data on models, where the amount of training data is increased by performing geometric transformations on the existing dataset to artificially ”create” more information and thus improved generalization. During the training process of our CNN, instead of feeding the exact preprocessed images as inputs, a combination of geometric transformations including rotations 8 to 8 degree, horizontal/vertical translations of 0 to 5 pixels, shearing from 0 to 0.3 rads and scaling from 0% to 8% were applied. Moreover, the creation of noisier data used during the training phase helps the classification model generalize better.

D. *Optimization Tricks*

1) Linear SVM: Due to the purpose of the linear SVM, optimization was not necessary on our baseline linear learning model.

2) Feedforward Neural Network: No optimization tricks were used during the hyper-parameter tuning of the Feed Forward Neural Network because manual grid search of parameters was implemented.

3) Convolutional Neural Network: In the case of CNNs, besides tuning parameters, optimization tricks were used in order to design a higher performance network. In fact, convolutional layers were fed into other convolutional layers, then pooling/subsampling layers were added.
As for the non-linear activation function, we used ReLU instead of sigmoid functions, which would optimize the network by reducing the likelihood of the gradient vanishing during the training process.
In addition, in order to accelerate training times, a callback function was used to reduce the learning rate, if a certain metric did not improve for a number of ”patience epochs”. We chose to reduce the learning rate by a factor of 2, if for 3 epochs, the accuracy of the model did not improve, with a minimum learning rate of 0.00001.

V. RESULTS

A. *Linear SVM*

The accuracy on training and validation sets for different values of C are shown in Figure 4. The best accuracy

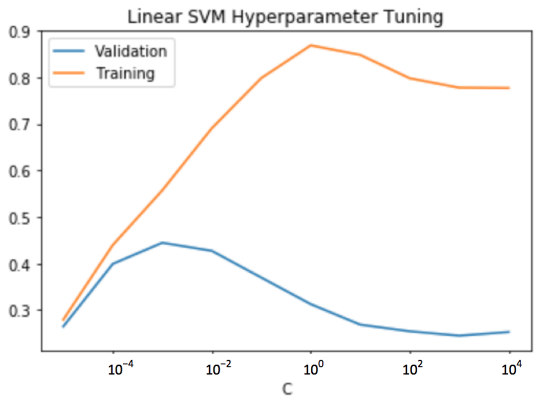

Fig. 4. Training and validation performance for different value of C

was achieved for $C = 0.001$. The resulting SVM classifier achieved an accuracy of 0.444 on the validation set. Clearly the best linear SVM classifier has overfitted the training data as we can see from Figure 4. 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.

B. Feedforward Neural Network

Using every combinations of learning rates and layer architectures to find the best model, we found that the best accuracy 51.5% on the validation data was obtained by the network composed of 2 hidden layer with 400 neurons for first layer and 300 for second layer and a learning rate of 0.01.

For all the networks we tried, we used the $sigmoid$ activation function. The main reason why we use sigmoid function is because it exists between 0 to 1. Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, $sigmoid$ is the right choice.

Figure 5 shows the accuracy for models with different neurons with one hidden layer. It shows that 400 neurons give the best result. Figure 6 shows the accuracy for models with different numbers of hidden layers. It shows that blindly increasing the number of layers cannot always improve model performance. Figure 7 shows the accuracy for models with different learning rates with one hidden layer and 400 neurons. The learning rate of 0.1 was too high and 0.001 was too low.

C. Convolutional Neural Network

Table I shows part of all models developed during the training and testing process. We have tried more than 20 models, for simplicity, we only included four models which can represent the improvement of the entire training process. As we can see decreasing the filter size can improve the model performance by near 5%.

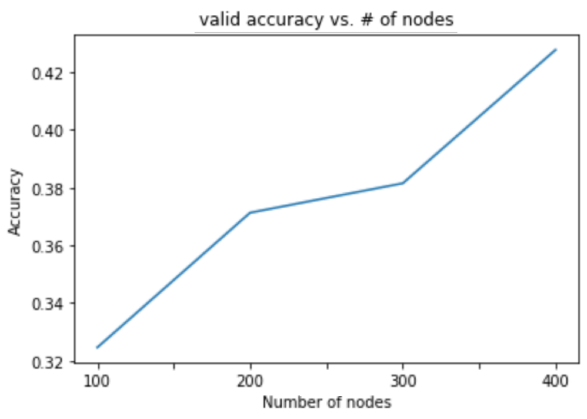

Fig. 5. Validation accuracy for different layer size

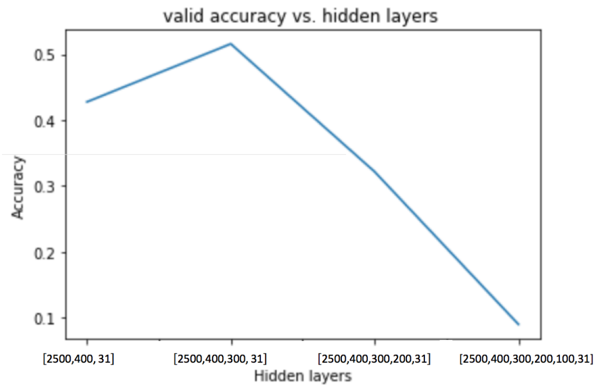

Fig. 6. Validation accuracy for different layer size

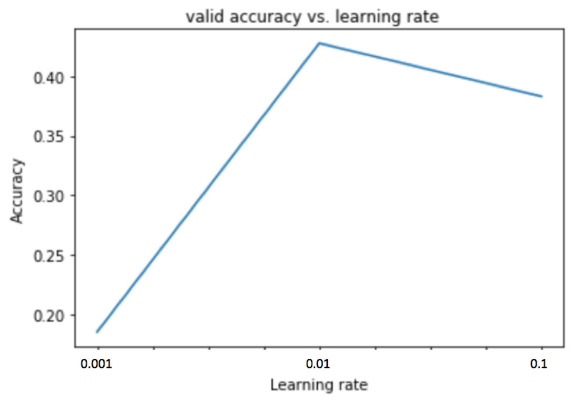

Fig. 7. Validation accuracy for different layer size

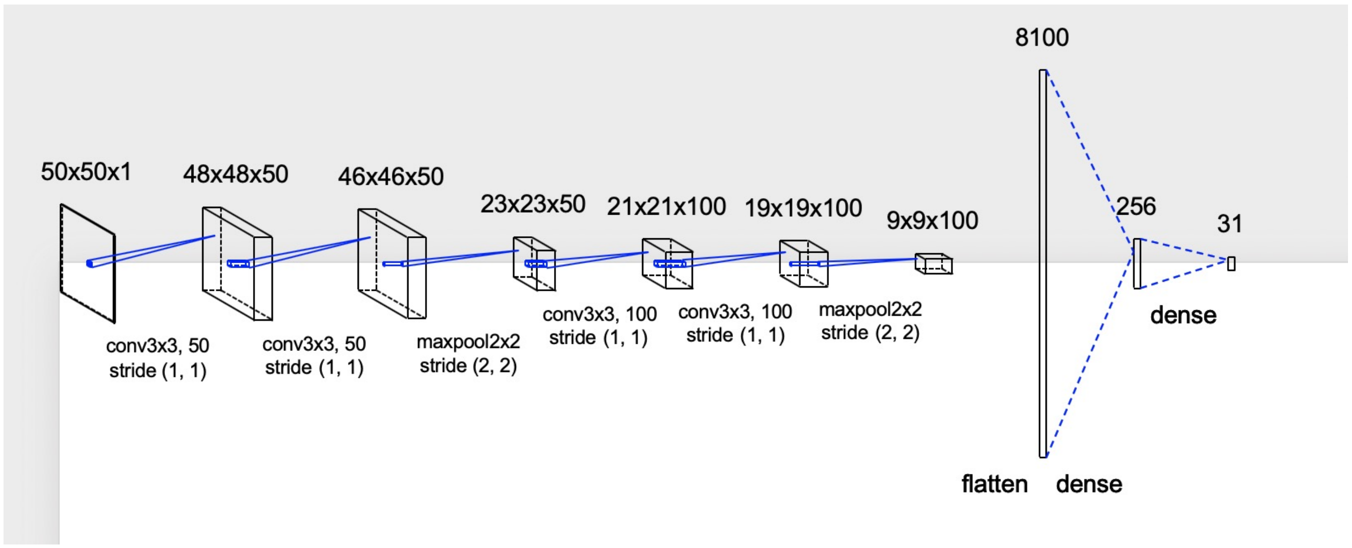

Fig. 8. Architecture of best CNN model

We started with very basic CNN model with 2 layers and no dropout, which achieved an accuracy of near 60%. Then, we discovered that failed predictions from the validation set were mainly objects with similar shapes, therefore, we concluded that the filter size might be too large in this case and we decided to decrease the filter size and increase the number of layers and add dropout following every two layers and before the last dense layer for model 2. And we decided to decrease the filter size for model 3. However, stacking 2 more layers did not give us better results, as we can see from model 4.

For the best model, as shown in Figure 8, there are 2 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 3 dropout regulations being applied, two of them 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, we believe this will prevent overfitting and reduce the interdependent learning amongst the neurons. From 10 and 11, we can see the our best model does not overfit and converges at near epoch 6. However, as we can see from Figure 9, squiggle and spoon are most wrongly classified by our model, which means that the model cannot clearly classify objects with relatively same shapes, like squiggle and moustache, and spoon and pencil.

The drawback of the model is its complexity, and the fact that it needs a lot of computational time to train. There is a trade off between complexity and accuracy.

VI. DISCUSSION

Table II gives a summary of all the classifiers discussed in the report.

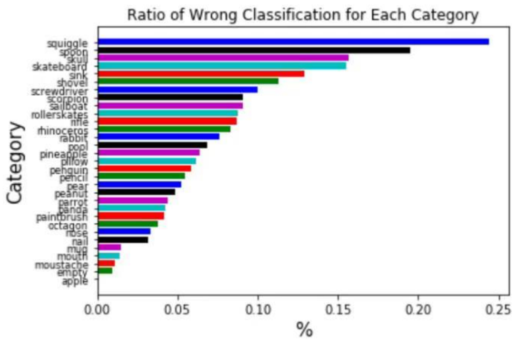

Fig. 9. Ratio of Wrong Classification for Each Category

TABLE I
VARIOUS CNN MODELS AND THEIR ACCURACY ON VALIDATION SET

model	layers	regulation	max pooling layer	filter size	accuracy
1	2	n/a	1	15,5	60.5
2	4	3 dropouts	2	5,3	75
3	4	3 dropouts	2	3,3	79.9
4	6	3 dropouts	3	9,5,3	79.1

A. *Methodology Advantages/Limitations*

1) Advantages: With systematic choice of hyperparamter values, we have achieve satisfied results from the three classifiers. The standardized tests easily helped visualize the importance of each parameter, especially in the case of the CNN. We mainly focused on building and tuning CNN model, since even basic CNN model performs much better than linear SVM and FFNN. However, for feed forward neural network, there are many hyperparamters to tune, and we were only able to test a few. In addition, the feature preprocessing done on the raw image data to remove noise and extract the hand-drawn object significantly improved classifier performance.

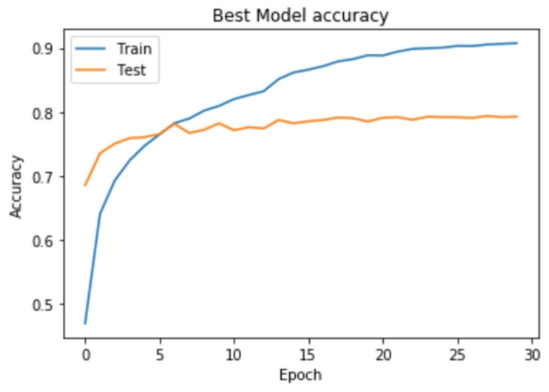

Fig. 10. Best Model Train and Valid Accuracy Plot

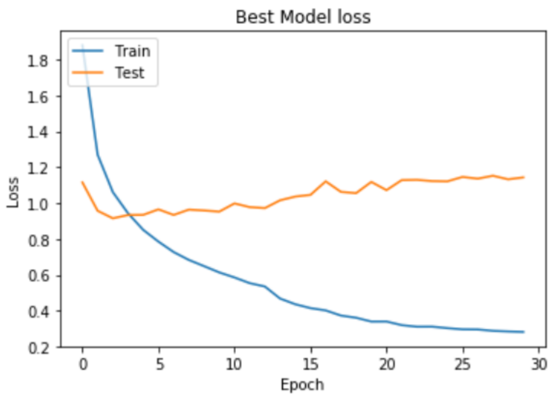

Fig. 11. Best Model Train and Valid Loss Plot

2) Limitations: However, since computational resources were limited, there is space for us to imporve our model. In addition, while looking for optimal hyperparameters, perhaps the range of values we searched on did not contain the best hyperparameter. Hence, we could improve our test methods by searching on a larger grid of test values. Also, the best model fails at classifying similar objects, like sink and pool. This means that there is also space for improvement on both model itself and preprocessed methods.

B. Areas of Future Work

1) Preprocessing Step: To better preprocess the raw images, we could try other popular techniques and make the preprocessed images more separable for the model to classify, for example, we could use a median filter on top of our method. And we could also write an algorithm implemented by hand instead of using opencv library to better filter out the noise and extract the hand-drawn object for this dataset specifically.

2) Optimization: There is not much we can do for linear SVM. Nonetheless, for FFNN, we could tune more hyperparameters and vary the number of neurons, number of layers, and activation function if we have more time.

Since this course is just about general introduction of machine learning and we only had one course for CNN, our knowledge in optimization could be improved, which would help us understand hyperparameters contained within the different gradient descent techniques. In fact, without further studies, we mostly used default values from keras library. In conclusion, to study more about deep learning could help us understand how to optimize network structure itself, like the number of filters needed per convolutional layer, the dropout rate needed, the subsampling layer, the number of hidden layers required, and so on.

TABLE II
PERFORMANCE OF DIFFERENT CLASSIFIERS FOR MODIFIED MNIST

Classifier	Parameters	Accuracy
Linear SVM	C=0.001	0.444
Feedforward NN	2 hidden layers, 400 and 300 hidden neurons, lr=0.01	0.515
CNN model 3	see Table I and Figure 8	0.799

STATEMENT OF CONTRIBUTIONS

The authors, Yutong Yan, Bridget Liu, and Yang Lu have all taken direct roles in the three classifier implementations, methodology and model evaluations, feature preprocessing, as well as the composition of the report.