Doodle Classification

Kaggle Competition — Team Bernoudles

He, Jingyi
260678541
jingyi.he@mail.mcgill.ca

Rahman, Aanika
260662187
aanika.rahman@mail.mcgill.ca

Tang, James
260685449
guokai.tang@mail.mcgill.ca

Zhao, Zhuoran
260716696
zhuoran.zhao@mail.mcgill.ca

***Abstract*—'Quick Draw' doodles dataset is a large hand drawn image dataset provided by Google. In this project, the subset of the 'Quick Draw' doodles dataset is analyzed. The goal of this project is to classify the hand drawn doodles into 31 classes. In this report, we show the outstanding performance of CNN against other models. We also show the careful preprocessing on images, proper features selection, and the model construction significantly improve the performance of the models.**

I. Introduction

Image analysis is a popular area of research in the domain of Machine Learning. Tackling a variant of the Google's 'Quick Draw!' dataset, the goal is to classify hand drawn images into 31 classes. The provided training and test datasets contain 10,000 labelled and unlabelled hand-drawn images respectively. Each image of size (100,100) consists of a doodle surrounded by randomly generated noise, where for one class labelled empty, each image consists of no doodle and only noise.

In order to maximize prediction accuracy, different machine learning algorithm methods from baseline to more complex models are tested, including SVM[1], neural network, k-NN, and CNNs.

We use OpenCV [1] for image preprocessing, and algorithms are built using scikit-learn [6] and tflearn [2].

II. Feature Design

The goal for this section is do the preprocessing to get rid of less important features which cannot provide the useful information for us to do the further training. Overall, this procedure can be separated into several parts.

A. Thresholding

Given the nature of the dataset, the first step in preprocessing entails removing noise from each image without altering the original doodle. Starting with image thresholding, we compare pixel values of the input image f with some threshold T and make a binary decision to output a binary image g. Where (i, j) represent the coordinates of the ij^{th} pixel, the decision is made for all i, j as follows:

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

Since the doodle and the noise share similar pixel values, binarizing the image according to a chosen threshold can either reduce noise ever so slightly or alter the drawing of interest. Hence, this step is intended to sharpen the image, as a stepping stone before extracting the largest connected component.

[1]scikit-learn

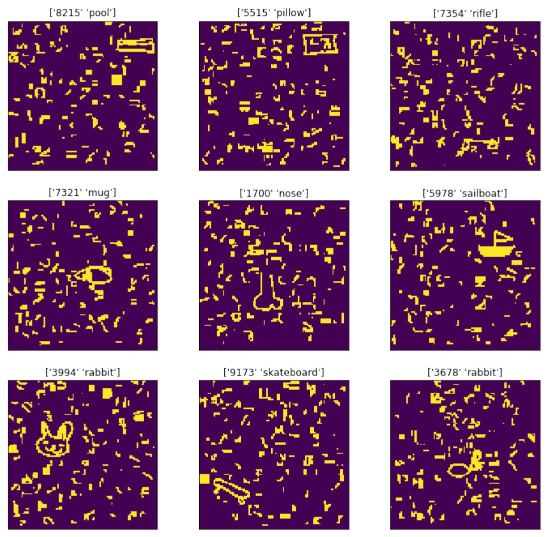


Fig. 1. Original Doodles before preprocessing

B. Denoising

Using the OpenCV function *connectedComponentsWithStats()*, the image is broken into a series of connected components with 8-way connectivity (i.e. pixels are grouped as a component if any of the eight neighboring pixels connect). Given the ability to check for size, all components except the largest component is interpreted as noise and thus removed, isolating the supposed doodle.

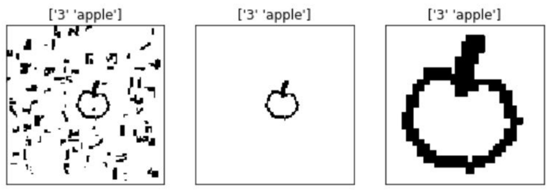


Fig. 2. from left to right: raw image, denoised image, cropped image

C. Cropping

The mentioned OpenCV function provides the starting coordinates and size of the bounding box surrounding the largest connected component. Since the bounding box is a rectangle with differing width/height, the greatest dimension is taken to be both width and height of a square image. Before finalizing the mask to crop this bounding box of interest, the image is further padded slightly. Once cropped, all images are resized to (35,35) and binarized once more as resizing introduces non-binary pixel values.

D. Augmentation

The final step in the pre-processing is to augment the image to add variances of each image as a supplement to the 10,000 images dataset provided. Image augmentation provides the model with more data to train. For our image augmentation, we generate randomly rotated images, mirrored images, randomly shifted images and for each of the rotated image, we randomly generate a piecewise-affine variation (locally distorted image) and a general-affine variation (globally distorted image). This step is only performed before carrying out the CNN algorithm.

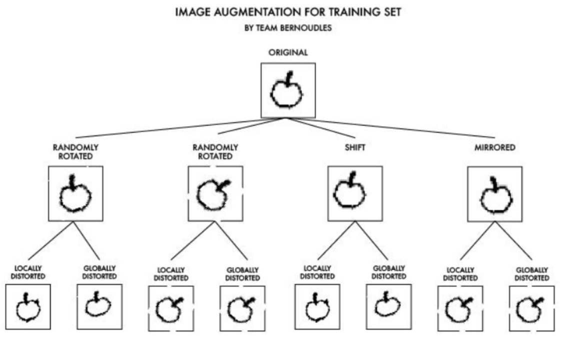


Fig. 3. Augmentation

III. Algorithms

A. Linear SVM

Support Vector Machine(SVM) is to construct a set of hyperplanes to separate the data points which are not separable in its lower dimensional space. [3] The goal of SVM model is to maximize the distance of the nearest point to the hyperplane, which is also known as the margin between classes. The SVM with linear kernel chooses one-vs-rest or one-vs-one for multi-class classification problem. In the SVM experiment we perform, the one-vs-rest strategy is used, which involves solving 31 binary classification problems. [5] In order to allow misclassifcation in the process, hyper-parameter penalty term C is introduced in the model. To fine-tune this hyper-parameter, we use 3 fold cross validation.

B. Feed-Forward Neural Network

The structure of a "feed forward" neural network is such that neurons at layer l become the inputs to the neurons at layer $l+1$ (i.e. forward pass), until the last layer which contains the predicted output. The features are utilized as neurons for the input layer and given a 31-class classification problem, the output layer contains 31 neurons.

In the forward pass, we obtain output from weight matrices and bias vectors following the equations below:

$$A = Input \times W + Bias \tag{2}$$

$$Output = activation(A) \tag{3}$$

The $tanh$ activation is used to calculate the activation for every hidden layer, restricting the output between 0 and 1. We choose to use the $sigmoid$ activation at the output layer due to its property of ignoring the minimal structure of the dataset. It is also a generative model, which provides good representation between the model and the individual features and responses.

$$sigmoid(A) = \frac{1}{1+e^{-A}} \tag{4}$$

After running a forward pass, we compute the correction for each weight using backpropagation. Using gradient descent methods and the simple notion of the chain rule to update neurons weights and biases, the following equations apply, where η is the learning rate:

$$W = W - \eta \cdot \frac{dW}{n} \tag{5}$$

$$Bias = Bias - \eta \cdot \frac{dBias}{n} \tag{6}$$

C. k-NN

k-NN is a non-parametric lazy classifier. The most computation tasks in this algorithm is in the classification step. The class of a given input is determined by the majority vote of the nearest K samples. The formula for calculating distance is also crucial for the k-NN model.

D. CNNs

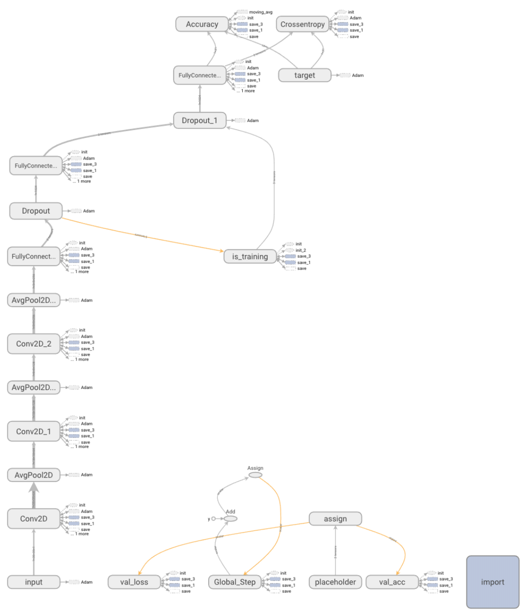


Fig. 4. CNN model

Our final model is Convolution Neural Network, which is one of the most popular models in computer vision for image classification. Different from Feed Forward Neural Networks, CNN model feeds in the input to the next layer in small patches, such that their local connectivity allow better feature detections, that include edges, corners. Some of those features are hard to detect in fully connected neural networks. We choose to introduce multiple hidden layers within the network. Hence, our CNN model consists of three convolutional layers, all of which are followed by average-pooling layers which perform a down sampling operation to reduce the variance of the model to reduce overfitting, and also acting as a filter to the image for better classification. [4] Finally two fully-connected layers with a final 31-way softmax are followed. We use a very efficient GPU implementation of the convolution operation to make training faster. To reduce overfitting in the fully-connected layers we set up the dropout rate and L2 regularization that proved to be very effective.

IV. Methodology

Using the proper loss function, in general we train our classifier as follows:

1) Perform the same preprocessing on both training and testing dataset as described in the feature design section.
2) Transform the text labels of training set to one-hot-encoding.
3) Split our 10,000 training images into training, validation and test set with ratio 8:1:1. For the Feed-forward Neural Network, other ratio splits are compared.
4) Reshape all samples to proper input size.
5) Use the training set to train the model and tune hyper-parameters base on the accuracy of the validation set. As cross validation is implemented for the Feed-forward Neural Network, tune hyper-parameters on the average accuracy of the validation set across all folds.
6) Evaluate the model base on the performance on test set.
7) Repeat step 4 and 5 to improve the model.
8) Use the best model to predict the final test set and make the submission.

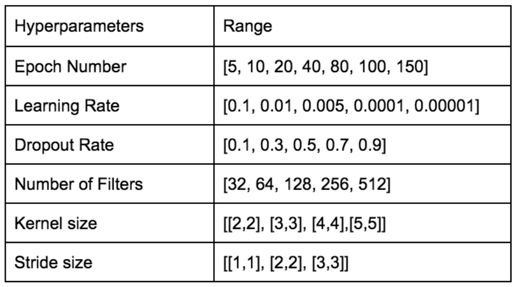
A. Hyper-parameter Tuning & Regularization

1) SVM: We use grid search for hyper-parameter tuning. To give a comparable result for the other model used in the later experiments, we use predefined validation set which is the same as in other model. We tested different penalty hyper-parameter values among 1e-9 $\sim$ 100.

2) k-NN: There are two hyper-parameters of interest, one is the distance calculation formula, the other one is the number of neighbours. We grid search on the training set and tune these two hyper parameters base on the performance on validation set. We tune k on the range(1,10), and p on Manhattan distance and Euclidean distance.

3) NN: Since the neural network is computationally expensive to train, a simple architecture is maintained. As the hyper-parameter in focus, the different values for the number of nodes per hidden layer tested include (250, 300, 350, 400, 450, 500, 550, 600). Although the number of hidden layers and learning rate is also explored briefly, 1 hidden layer and a learning rate of 0.5 is maintained for the majority of testing.

4) CNN: For augmentation part, we first use tensorflow do images augmentation by adding random vertical and horizontal flips, and random rotations with max 30 degrees of angle. For the tunning part, in order to maximize accuracy on test set, we use the CNN model with several hyper-parameters that include epoch number, learning rate, dropout rate, number of filters, kernel sizes and stride sizes. The corresponding range is listed below. Besides these hyper parameters, we also tries batch normalization after each convnet layer sets and different number of layers as the original models to tune those parameters. Then we use the validation set accuracy as a performance measure, we select the best parameters using grid search on all the hyper-parameters proposed.



Hyperparameters	Range
Epoch Number	[5, 10, 20, 40, 80, 100, 150]
Learning Rate	[0.1, 0.01, 0.005, 0.0001, 0.00001]
Dropout Rate	[0.1, 0.3, 0.5, 0.7, 0.9]
Number of Filters	[32, 64, 128, 256, 512]
Kernel size	[[2,2], [3,3], [4,4],[5,5]]
Stride size	[[1,1], [2,2], [3,3]]

Fig. 5. CNN's hyper parameters

V. Results

As our baseline classifier, linear SVM achieves 0.399 accuracy with the best hyper-parameter C equal to 1e-7. The fully connected feed forward neural network model with 0.28. Compared to these two classifiers, the k-NN and CNN model perform much better. Here are the detailed results for each model.

A. SVM

The result of Linear SVM has a worse performance on all training set, validation set and test set comparing to the complex CNN models. The best accuracy on validation/train set obtained by Linear SVM is 0.639 with C = 4, but the accuracy on test set is only 0.216. This is because the model is refitted on the whole train and validation set after grid search by default. Therefore, the best hyperparameter is chosen based on the accuracy score on test set, which is 1e-7 with test accuracy 0.399.

The accuracy of validation set shows that the value of C should be chosen from small values, but it cannot be as small as 1e-9. The results make sense since there are 31 classes in our experiment, and the objective function of SVM is to maximize the margin between classes. The penalty term gives a soft margin for misclassification but in the meanwhile it should prevent too many misclassifications. Thus, C should neither be too large nor too small.

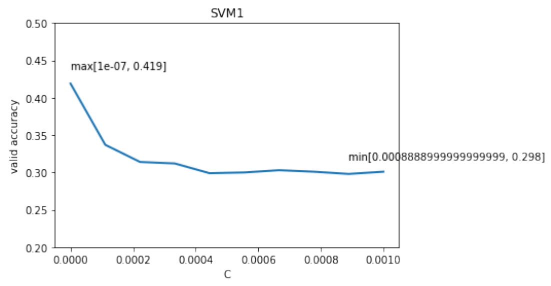


Fig. 6. Accuracy vs penalty C for model SVM

B. NN

After testing various combinations of learning rates and layer architectures to find the best model, we find that the best accuracy on the validation data is obtained by the network composed of 1 hidden layer with 500 neurons and a learning rate of 0.5. Under cross validation, this architecture resulted in a average validation accuracy of 0.29, followed by a test accuracy of.

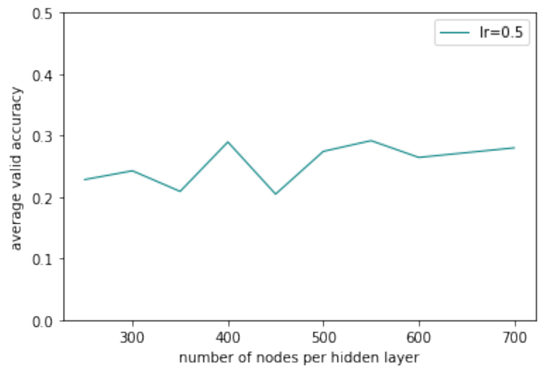


Fig. 7. Number of nodes vs. validation accuracy

When considering which learning rate is most appropriate, learning rates below the chosen 0.5 gives mixed outcomes. Similarly, when informally exploring if an extra hidden layer improves performance at a learning rate of 0.5, the average validation accuracy halves in some cases. Furthermore, although results are not formally noted, the network performs better with a 2:6:2 ratio split instead of the usual 8:1:1 ratio split. This is perhaps due to the tendency for the network to overfit with more training data.

C. k-NN

k-NN achieves the best result when k=6 using Manhattan distance.(Fig. 7) Notice that the accuracy does not fluctuate much as the number of neighbors increases. Interestingly the accuracy when k = 1 is not as low as we expected, as discussed in class small neighbor would generally leads to overfit. The accuracy on training set is 1 when k=1, which is obviously an overfitting, and decreases as the number of neighbor increases.

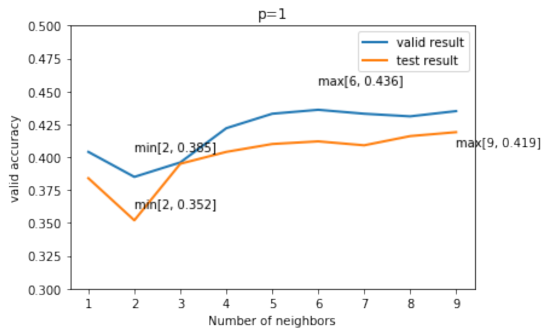


Fig. 8. Number of neighbors vs accuracy using euclidean distance

D. CNN

Among all four models we test, CNN model has the best performance without tuning with a validation accuracy of 73%. Upon the addition of several layers and fine tunning hyper parameters, we successfully obtain an accuracy of 78.4%. We finally choose our CNN model with number of filters[64, 128, 256], kernel size [2,2] for first layer and [3,3] for all other layers . For the epoch number, as we can observe in the graphs, after epoch 80, the accuracy does not increase but goes down. This indicates training more epochs would lead increasing on training accuracy but overfitting on validation accuracy. Besides, it is found that 60 epochs gave the best accuracy.

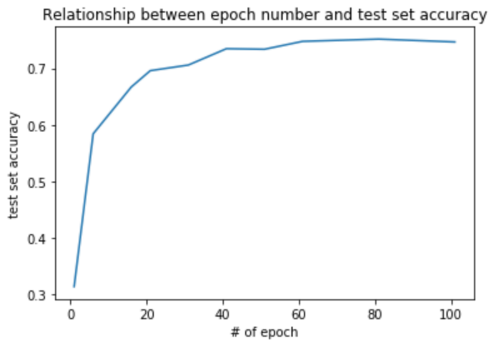


Fig. 9. Epoch number vs Accuracy for model CNN

The Dropout Rate does not seem that sensitive to the accuracy when it is greater than 0.3. With fine-tuning the dropout rate from 0.1 to 0.9, and using the learning rate with 1e-3 and 60 epochs, the dropout of 0.5 is found to best regularize a model.

However, different learning rates do affect the speed for growth accuracy a lot. We choose several values for the learning rate that varies from 1e-1 to 5e-4 and keep the other parameters unchanged(60 epochs, 0.5 dropout rate). Then it turns out that the learning rate with 5e-4 yields the best accuracy. Finally, for the batch normalization, we add it after each convnet layers, but this turns out do not significantly impact the validation accuracy.

Dropout rate	Validation Set Accuracy
0.1	0.552
0.3	0.739
0.5	0.743
0.7	0.742
0.9	0.725

Fig. 10. Dropout Rate VS Accuracy for model CNN

Learning Rate	Accuracy
1e-1	0.037
1e-2	0.045
1e-3	0.721
1e-4	0.743
5e-4	0.747

Fig. 11. Learning Rate VS Accuracy for model CNN

VI. Discussion

A. SVM

Each image (100,100) with 10,000 pixels in total can be viewed as a flat sequence of input features which disregards the relative 2D spatial location of pixels. This would cause a problem if the doodles are not in the same position on canvas for all doodles. Thus, working with doodles that are all centered in the preprocessing process step does improve the performance of all models. Moreover, SVM model is easier to train in comparison to CNN, and it has less hyper-parameters to tune. However, one of the cons of the Linear SVM model is that it is a linear model with linear decision boundary. The decision boundary for all classes versus the others are not necessarily linear. In the meanwhile, the complexity of the model is not enough to describe the differences across classes.

B. k-NN

k-NN classifier generally would not be used for image classification problem in application since the curse of dimensionality. Further notice that the computation on test set is very expensive especially when the dimension of input is large. k-NN uses all the features to compute the distance equally, which means if the features of input are sparse, the percentage of misclassification would be high. But one of the advantage of k-NN is that the decision of an input is only dependent on the small neighbor of similar objects.

C. NN

The performance achieved by our neural network model is far from ideal and is unexpectedly lower than our SVM performance. There are many parameters to be set in a neural network and optimizing the network can be challenging, especially to avoid overtraining.

D. CNN

Among all the classifiers, CNN model has the best performance with best accuracy 0.784 for test set. Different from the NN model, the neurons in CNN are arranged in 3 dimensions that are width, height and depth. And each layer transforms an input 3D volume to an output 3D volume with some differentiable functions that may or may not have parameters. These features of CNN allow it to take advantage of the two-dimensional structure of the input data, and also allows the CNN to recognize specific shapes in the images, such as circle and curve. This special characteristic of CNN allows it to perform better for image classification problems comparing to other neural networks and baseline classifiers.

The weakness of CNNs lays in the amount of data we have. CNNs have a large number of parameters, and for our image classification problem, we only have a small training set to work with. With the small data-set, we often over-fits the model during training. Since we only have 10,000 training data but 31 classes in total, CNN did not perform as good as we expected.

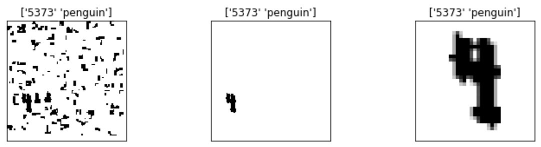
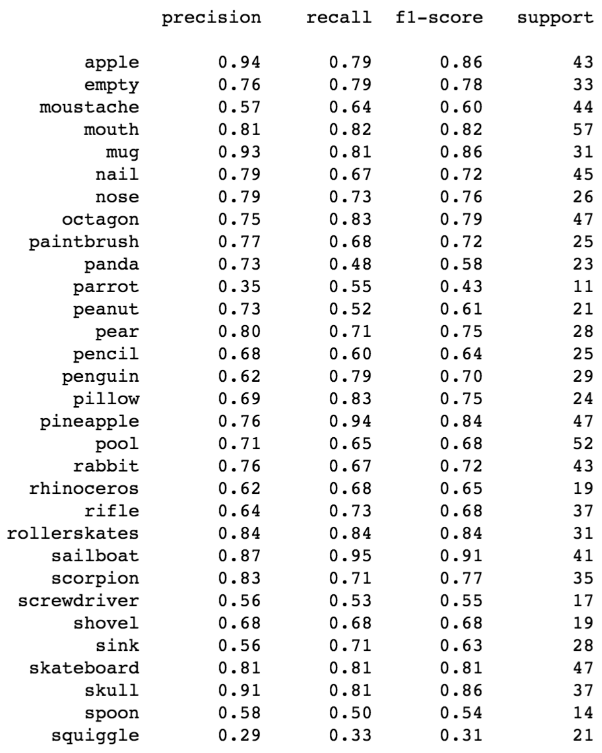


Fig. 12. From left to right, Original Image, Image after Noise reduction, Image after crop

Also for the specific classes such as penguin shown in the figures above, after cropping, it becomes almost unidentifiable because all the nearby pixels become connected. This makes it difficult for our CNN model to identify patterns from this class. From figure 13 below, we observe that almost all pixelated classes have much lower f1-score and precision, like class ’squiggle’, ’penguin’ and ’spoon’.

E. General

The classification report shows the f1-score for each class is in the range of 0.6 to 0.9 except for class squiggle, parrot, and class panda. And from the classes distribution, it turns out that the number images for the three classes mentioned above are much lower than other classes. This explains the reason why these classes perform worse. As the patterns of class ’squiggle’ are similar to class mustache or class empty, it is easy to be misclassified. Then we infer the bad performance can be



```
              precision    recall  f1-score   support

       apple       0.94      0.79      0.86        43
       empty       0.76      0.79      0.78        33
   moustache       0.57      0.64      0.60        44
       mouth       0.81      0.82      0.82        57
         mug       0.93      0.81      0.86        31
        nail       0.79      0.67      0.72        45
        nose       0.79      0.73      0.76        26
     octagon       0.75      0.83      0.79        47
  paintbrush       0.77      0.68      0.72        25
       panda       0.73      0.48      0.58        23
      parrot       0.35      0.55      0.43        11
      peanut       0.73      0.52      0.61        21
        pear       0.80      0.71      0.75        28
      pencil       0.68      0.60      0.64        25
     penguin       0.62      0.79      0.70        29
      pillow       0.69      0.83      0.75        24
   pineapple       0.76      0.94      0.84        47
        pool       0.71      0.65      0.68        52
      rabbit       0.76      0.67      0.72        43
  rhinoceros       0.62      0.68      0.65        19
       rifle       0.64      0.73      0.68        37
rollerskates       0.84      0.84      0.84        31
     sailboat       0.87      0.95      0.91        41
    scorpion       0.83      0.71      0.77        35
 screwdriver       0.56      0.53      0.55        17
      shovel       0.68      0.68      0.68        19
        sink       0.56      0.71      0.63        28
  skateboard       0.81      0.81      0.81        47
       skull       0.91      0.81      0.86        37
       spoon       0.58      0.50      0.54        14
    squiggle       0.29      0.33      0.31        21
```

Fig. 13. Accuracy for Each Class

reasoned by this. In addition, since images from class parrot, and panda are more complicated than others, we can also improve our model by using a deeper CNN model (with more than six layers), with the hope of extracting more features from doodles.

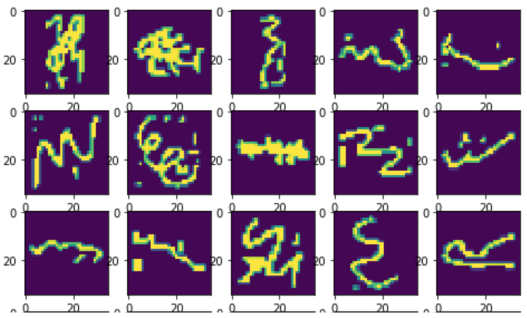


Fig. 14. Squiggle Figures

F. *Areas of further work*

Further work can be done to improve the accuracy. For the preprocessing part, the images such as squiggles can be improved by recognizing some specific patterns. Also, the images' distinguishability can be improved, the lines and patterns inside the images can be recognized better thus the accuracy could also be improved. To avoid losing the important part in the images, there are also some data augmentation techniques such as 'random crop', 'random denoise' or 'add gaussion noise' can be used for further experiment. Like discussed above, all of the models can benefit from adding sharpening to the preprocessing as well.

For the SVM model, we suggest to explore the nonlinear kernel trick to increase the nonlinearity of the model to better accommodate the image classification problem.

Since Neural networks are expected to be less sensitive to noise than statistical regression models, perhaps testing the network with less preprocessing may result in better accuracy. Furthermore, as the impact of the number of hidden layers and choice of activation functions can be more thoroughly explored, there is room for improvement.

For the CNN model, we can try add batch size as another hyper parameters to train. Since training neural networks in batch helps reduce memory and reduce training time. We can also try a deep CNN by adding more layers to the model. Furthermore, it is worthwhile to perform an extended hyper-parameter tuning on the CNN model. Since CNN has many hyper-parameters like filter size and kernel size for convolution layers, stride for pooling layers and number of neurons for each fully connected layers. It is difficult to find the best set of hyper-parameters for our model given the time-frame of this project. But it is indeed likely to find a better set of hyper-parameters if we extend our range of selection. And the CNN model can also be improved by implementing a "sharpen' procedure into our preprocessing process where each cropped drawing undergoes a filter that sharpens the image and reduce pixel density without changing the image. Unsurprisingly. this filter can be implemented using CNN as well, because of CNN's ability solve image-related machine learning problems. Alternatively, we can perform transfer learning with the pre-existing image CNN models and tweak the model in some way such that it fits the need of our task.

Another technique that is worth to test out is ensemble modeling. That is, the dataset is trained on different models and then use boosting or bagging to combine the result from each model to give a overall prediction on the input data. Ensemble modeling also prevents the model from overfitting.

VII. Contribution

The authors, Jingyi He, Aanika Rahman and Zhuoran Zhao, have all taken direct roles in the all classifier implementations, methodology and model evaluations, feature preprocessing, as well as the composition of the report.

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

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Aymeric Damien et al. Tflearn. https://github.com/tflearn/tflearn, 2016.

[3] Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.

[4] Karpathy, Andrej. Cs231n convolutional neural networks for visual recognition, 2017. http://cs231n.stanford.edu/2017/, Last accessed on 2018-11-26.

[5] Farid Melgani and Lorenzo Bruzzone. Classification of hyperspectral remote sensing images with support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*, 42:1778–1790, 2004.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.