COMP551 Kaggle Competition Report
Team: RedWolves

Dmitrii Tiron
260654807
dmitrii.tiron@mail.mcgill.ca

Tan Khoa Gael Duong
260691624
tan.k.duong@mail.mcgill.ca

Richard Hughes
260708750
richard.hughes@mail.mcgill.ca

1 Introduction

The Doodle Recognition task consists of developing an image classification algorithm that is able to appropriately classify hand-drawn images into one of 31 classes of images. The raw images also include a lot of noise and the target drawings were placed in random locations within the image. We created a pixel-based feature set, which was optimized using heuristic methods. We then trained multiple classifier models using 3 different learning algorithms. We found that the best performance was achieved when the optimized feature set was fed to the custom CNN learning algorithm.

2 Feature Design

Given the raw images with significant amounts of random noise, we decided to use the OpenCV python library for faster clean up.

2.1 Noise reduction

In order to get rid of artificially added noise in form of random shapes, we decided to get rid of all shapes except the largest shape. To pick the largest shape the first step is to find all the shapes in a given image, which we did using $findContours$ function. We then choose between two shapes: the shape with highest contour area (as computed by $cv2.contourArea$) and the shape with the highest contour perimeter (as computed by $cv2.arcLength$). Next, we computed the bounding rectangles of those two contours and picked the largest shape to be the shape with the largest bounding rectangle area. The motivation for this strategy was experimental: we noticed that, for some types of shapes, the target drawing is better defined using the contour perimeter rather than contour area. This approach improved our performance by up to 2% when compared to the strategy of always picking the shape with largest contour area.

After picking the largest shape, we generated an image and set the intensity value of the pixels of the selected contour and the space it encloses to 1. We then executed an element-wise multiplication between the image that only has the selected shape and the raw image. The image that is the result of this multiplication will preserve the initial intensities of the selected shape and of the elements inside, while the intensity of all pixels outside of it are set to 0.

2.2 Feature reduction

Given that the target drawings seemed to us to be way smaller than the actual size of the image, we decided to reduce the number of features by placing the target image in the middle of the smallest rectangle that can enclose any target image from the train and test set. So we scanned all the images in the train and test set and, for each of them, selected the target shape using the method described in 2.1. We then recorded the size of the largest bounding rectangle from all those images. This size was used to generate the new images of size 38x38 px from the initial train and test set images. Our next challenge was to properly include the target shape into the final image. For the sake of alignment of the drawings in the train and test set, we computed for each image, the mass center of the selected shape and moved the shape so that the mass center of the shape would end up in the center of the final image. The new sets of images were used as the actual train and test sets. This dimensionality reduction technique allowed us to reduce the number of redundant, empty dimensions, which speeded up our training time and reduced the model complexity. The number of features decreased from $10,000$ (100x100) to $1,444$ (38x38).

The intensity values for each pixel in each image were then normalized to be within the $[0,1]$ range.

3 Algorithms

3.1 SVM

We used the sci-kit learn implementation of Support Vector Machine algorithm with linear kernel as our first baseline algorithm. The algorithm attempts to find the optimal position for the separating hyperplane by updating its posi-

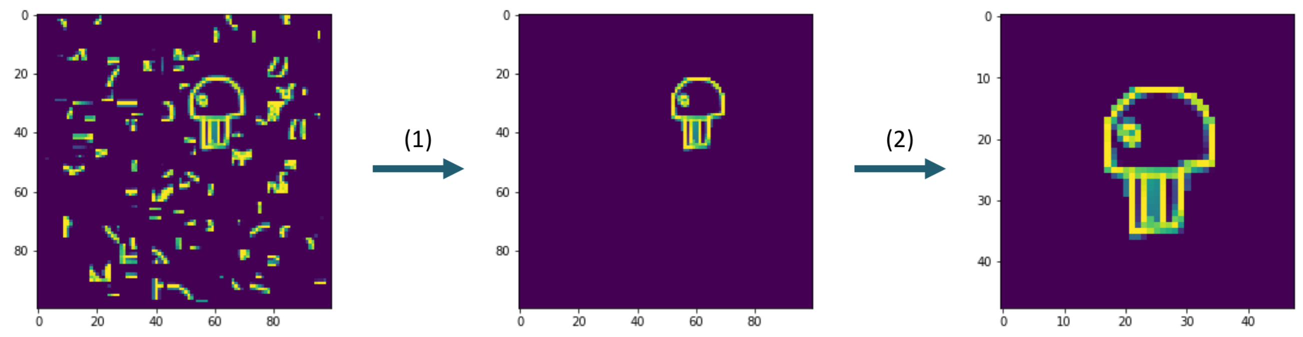


Fig. 1. Example of the result of application of our pre-processing steps. Process in (1) is the Noise Reduction step described in 2.1. Observe that all that is left is the target shape, which is to be classified. Process in (2) is the Feature Reduction step, where we center the target shape in the middle of a smaller image, reducing number of features and assisting the uniformity of data. (training set, example 255)

tion for as long as it improves classification accuracy. This hyperplane is then used to classify new data.

3.2 Neural Net

A fully connected feed-forward neural net with backpropagation was implemented from scratch as our second algorithm. Neural Net contains an input layer, an output layer, at least one hidden layer of neurons and edges that connect neurons from different layers. All edges have weights associated with them. A fully connected net contains edges that extend from each neuron of a given layer to every neuron of the next layer. The algorithm is updating the weights on the edges using backpropagation on every epoch as to minimize the value of the error function. The net with optimized weights is then used to classify new data.

3.3 Convolutional Neural Net

A convolutional neural network takes an image (2D array of pixels) as input and passes it to a sequence of convolutional layers to extract unique features of the image from low-level elements (edges, curves, lines) to high-level elements (forms,shapes). To reduce dimensionality of the filtered images CNN uses max pooling layers which reduce a close neighborhood of pixels to a single value in the reduced image. Individual pixels of the reduced filtered images are then flattened out into a vector be fed into multiple fully connected (dense) layers, which are then trained to output the classification labels as a vector of probabilities using softmax function. The training of this network implies optimizing the weights on the edges using backpropagation based on the error between the actual output and and the true output. This process is repeated on every examples from the training set k times (k = number of epochs), continuously evaluating the model accuracy on validation set and updating the weights to obtain the optimized network. We then use these convolutional, max pooling and dense layers with optimal settings to classify new data.

4 Methodology

4.1 SVM

Since we used the sci-kit learn implementation of the SVM algorithm, we were able to tweak some parameters to optimize the output. The model was selection was based on 5-fold cross-validation output. We optimized 3 hyperparameters: tolerance for stopping criteria, regularization strength and maximum iteration allowance. The tolerance for stopping criteria is the threshold for stopping the learning, in other words, its the minimal value of improvement for the algorithm to keep learning. Regularization strength parameter regulates how much misclassification error the algorithm will tolerate, necessary for the trade-off between overfitting and underfitting. Maximum iteration allowance is the maximum number of iterations that the algorithm is allowed to complete.

4.2 Neural Net

First the training data was split into 5-folds for hyper parameter tuning. Each of the training labels (numbers) were converted to binary vectors of size 31. For example 4 would be converted to [0,0,0,0,1,0,,0]. The output of each hidden layer node was transformed using the sigmoid function, transforming the output value non-linearly and keeping it between 0 and 1. When initially setting the weights, we used a random number between 0 and 1 but found that setting it randomly between -1 and 1 led to quicker and more reliable convergence. The weights were updated through stochastic gradient descent (updated after each sample) and the feed-forward and backpropagation calculations were carried out through matrix multiplication. The hyper parameters considered were number of layers, number of nodes and learning rate, determined by cross-validation.

Number of layers For number of layers, we found that as you increase the number of layers, the validation set accuracy increased, so we tried running the Neural Network with 1, 2, 3, 4, and 5 layers. More layers allows the network to have a greater impact on the output without having to increase the number of nodes in the hidden layers.

Learning Rates We next tried different learning rates of .001, .01, .1, and 1. Smaller learning rates can take longer to converge, but can help to ensure that a proper minimum is not missed. If the learning rate is too large, it could overshoot the minimum and begin to diverge.

Number of Nodes Finally, we tried different numbers of nodes in the hidden layers, beginning with a relatively small number of 50 and eventually increasing to 500. Similar to the effect of increasing the number of layers, augmenting the number of nodes in each hidden layer allows to extract more combined feature information. Higher volume of extracted information has a more significant effect on the output, but having too many nodes can result in overfitting to the training set.

4.3 Convolutional Neural Net

First, the data was split into a training set of 8000 samples and a validation set of 2000 samples. Each of the training samples was then reshaped into a matrix of size 38 x 38, which will be the input layer of the CNN. Each of the training labels (a number) was converted into a binary class vector (eg: $2 \rightarrow [0, 0, 1, 0, ...0]$, which will be the format of the output layer of CNN.

Network Architecture To select the best network architecture, we started with a simple Convo-Pool-Dense set up and then tried different types/numbers/orders of layers to optimize the feature extraction step. Our strategy was to keep adding more layers until the accuracy stopped improving. The intuition is to extract as many features as possible without creating an overfitted set up, where the algorithm extracts features that are strictly specific to the images from the training set. Of course, performance was also dependent on the depth, which corresponds to the number of filters at each convolutional layer. This parameter reflects the number of features that are extracted at a given layer from the input image. In our experimental set up, we gradually increased the value of depth from 64 to 128. We also optimized features like kernel size (size of the filter matrices), pool size (the size of the region grouped together in the max pool layers) and stride (step-size of the convolution & max pool layers).

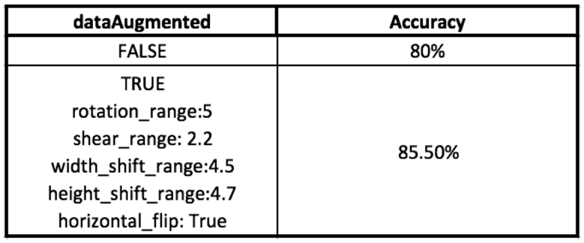
It was necessary to add max pooling layers to reduce dimensionality and allow for a more flexible model. This generalization works by creating a feature reflecting the values of a region of image instead of having features represent values of specific pixels. Naturally, this dimensionality reduction also speeds up the computations.

Finally, we added a fully connected layer, which is the neural net that will actually perform the classification. Implementing 2 hidden layers was the most reasonable decision from both the point of view of computational resources and performance-wise.

It is also important to note that when we apply backpropagation, the values are getting squished between 0 and 1 (for sigmoid function). As the algorithm learns, we keep multiplying these small numbers together, creating the vanishing gradient problem, so we added reLU activation function to alleviate this problem.

Learning optimization We trained the model with different learning rates (optimizer = Adam). Our goal here was to find the optimal trade-off between faster convergence (achieved with higher learning rates) and more accurate classification (achieved with lower learning rates). We also optimized the batch size, which defines the number of examples that is fed to the model before the weights are updated. As for the number of epochs, we simply increased the number of epochs until the validation accuracy stopped increasing. To prevent the model from overfitting, we applied dropout regularization technique between layers, which works by dropping some neurons of the layer. We tried different dropout values (0.1, 0.25, 0.5) and we observed that it is necessary to increase the drop out values as the layer gets bigger.

Training dataset augmentation Any given image that we learn to recognize will have a lower probability of being properly classified if its not aligned in the same way as the images in the train set. So we decided to inflate our train set by taking every image from the preprocessed train set and create new images with modified shapes. For each image we created 100 modified duplicates, where the modifications included rotation, shifting, shear and flipping horizontally. Each duplicate was modified in each of these ways by a random amount; the ranges of the values for modifications are reported in Table 1. We now obtained a train set that is 100 times the size of the initial train set, which now included modified shapes. This training dataset augmentation technique improved the accuracy of the CNN predictions by up to 5%.



dataAugmented	Accuracy
FALSE	80%
TRUE rotation_range:5 shear_range: 2.2 width_shift_range:4.5 height_shift_range:4.7 horizontal_flip: True	85.50%

Table 1. Reporting performance of the CNN model with and without the data augmentation. Accuracy: based on validation set of 2000 samples. Maximum settings for duplicate modification: [rotation_range: rotation angle in degrees; shear_range: shear angle; width_shift_range: horizontal shift in pixels; height_shift_range: vertical shift in pixels; horizontal_flip: horizontal flip allowed]

5 Results

5.1 SVM

For the SVM algorithm we observe that the hyperparameter that defines the best performing models is regularisation strength (C). Out of the top 10 models, top 7 results were all obtained by models with the value of C equal to 0.5, while the other two variables vary a lot. We notice that the other two values, tolerance of the stopping criteria (tol) and maximum iterations allowance (max_iter), have a less significant effect and are interrelated. It makes sense because given a certain value for tol and C there is a set number of iterations the algorithm will go through before stopping organically, so stopping the algorithm prematurely could have a negative effect on the performance. Thus, the max_iter value needs to be set in a way that does not to interrupt the training too prematurely, but also stops the training before the algorithm overfits to the training data. The best accuracy (average of 5-fold cross-validation) of 0.3432 was achieved with an SVM model with the following hyperparameter settings: C = 0.5, tol = 0.001 and max_iter = 100. When uploaded to Kaggle, the accuracy went up to 0.356 for the 30% of the test data available on the platform.

Rank	Score	C	max_iter	tol
1	0.3432	0.5	100	0.001
2	0.3421	0.5	100	0.00001
3	0.3394	0.5	100	1
4	0.338	0.5	500	0.01
5	0.338	0.5	1000	0.1
6	0.3376	0.5	500	0.0001
7	0.3369	0.5	1000	0.0001
8	0.3345	0.75	100	0.00001
9	0.3341	0.75	100	0.1
10	0.3306	0.75	100	0.001

Table 2. Reporting performance of SVM model with various hyper-parameter settings. Score: average score from 5-fold cross validation; C: Regularisation Strength; max_iter: maximum iterations allowance; tol: tolerance for the stopping criteria

5.2 Neural Net

As previously mentioned, tuning of the Neural Net involved adjustment of 3 hyperparameters: number of hidden layers, number of nodes in each hidden layer and the learning rate. In the case of the NN, our results clearly highlight the model that has by far the best performance (0.06 advantage over the second best), which was achieved with 1 hidden layer with 100 nodes and 0.1 learning rate.

We think that because of our pre-processing and powerful feature reduction strategies, we do not need more than 100 nodes to extract the most important features from the images. Consequently, having more than 1 layer or more than 100 nodes seems to result in overfitting the model to our training dataset, extracting combinations of features that wont generalize well to previously unseen images. The learning rate clearly has a very significant effect on performance as well, as we can see in the Table 3, this is the only factor differentiating the best and second best model. As expected, a smaller learning rate yields higher accuracy.

Rank	Score	Num_Nodes	Num_Layers	Learning_Rate
1	0.5215	100	1	0.1
2	0.4635	100	1	1
3	0.44	100	2	0.1
4	0.4385	50	1	1
5	0.428	100	3	0.1

Table 3. Reporting performance of NN with various hyper parameters. Score: the average accuracy from 5-fold cross validation; Num_Nodes: the number of nodes in each hidden layer; Num_Layers: the number of hidden layers; Learning_Rate: the value multiplied by the change in each weight during gradient descent

In the end, our most successful model had a 0.5215 accuracy score when evaluated with 5-fold cross-validation on the training set, and obtained an accuracy score of 0.482, when tested on the part of the test set available on Kaggle.

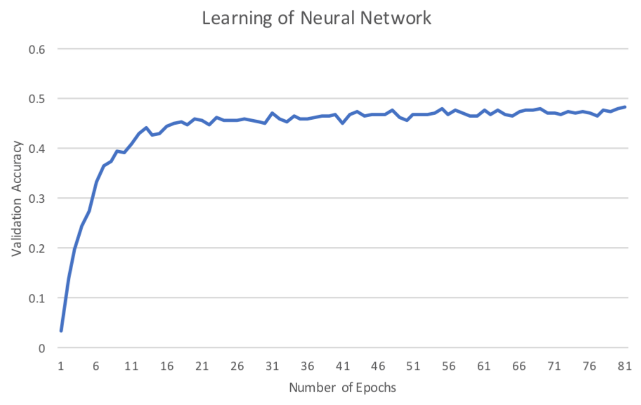


Fig. 2. Showing the performance improvement of a 1 layered Neural Network with 100 nodes and a 0.1 learning rate over the number of epochs.

5.3 Convolutional Neural Net

The third algorithm that we used required a lot of hyper-parameter tuning. After experimenting with multiple setups involving various numbers and ordering of the convolutional and pooling layers, we found that the best set up is to have 6 convolutional layers and 2 max pooling layers (Conv Conv Conv Pool Conv Conv Conv Pool). To obtain the optimal convolutional layers, we kept the depth between 92 and 128 and set both the kernel size and the stride to (2,2). As for the max pool layers,we found that the ideal pool size to reduce the filtered images without losing crucial information is (2,2) and the optimal stride is again (2,2).

Finally, we optimized the parameters of the dense layer, which interprets the extracted features to classify the image.The optimal learning rate for our dense layer with 2 hidden layers turned out to be 0.001. To find the optimal maximal batch size we tried 4 values ranging from 32 to 128 and found that going past 100 is hurting the performance, so we stopped there. The number of nodes in each hidden layer was set by increasing the number of nodes while it had a positive effect of accuracy. We found that the algorithm tends to converge when the number of epochs approaches ˜170-220. Finally, we found that the best thresholds for dropout regularization are 0.5 for big layers and 0.2 for small layers.

We can clearly see from the table reporting CNN results that the Network architecture and the learning rate of the dense layer had more significant impact on the performance of the algorithm, but it also looks like cutting off the learning prematurely by limiting the number of epochs, also has a negative effect on classification accuracy.

The best validation accuracy we obtained was 85.5% with the optimal CNN architecture of 6 convolutional layers (92 100 128 92 100 129 being the depth at each convolutional layer, respectively). Our predictions obtained an accuracy score of 0.844, when tested on the part of the test set available on Kaggle.

Rank	Score	Architecture	Epoch	Learning rate
1	0.855	92C 100C 128C P 92C 100C 127C P 512FC 31FC	200	0.001
2	0.844	64C 64C 64CP 64C 64C 64C P 512FC 31FC	200	0.001
3	0.832	92C 100C 128C P 92C 100C 127C P 512FC 31FC	100	0.001
4	0.823	92C 100C 128C P 92C 100C 127C P 512FC 31FC	200	0.0001
5	0.793	92C 100C 128C P 92C 100C 127C P 512FC 31FC	200	0.1

Table 4. Reporting performance of CNN with various hyper parameters. Notation: C: convolutional layer, P: pool layer, FC: fully connected layer, the numbers represent the depths (number of filters) at each convolutional layer, e.g: 64C : 64 filters at convolutional layer. Our final mode: 6 convolutional layers, 2 max pool layers, 2 fully connected layers and 3 drop out layers in between

6 Discussion

6.1 Pre-processing

Our pre-processing strategy has a couple of drawbacks. For the given dataset it performs well, because there are very few shapes that do not share the same contour. Imagine a roller blade and one of its wheels is not touching the contour of the boot. In this example, our preprocessing strategy will simply treat the disconnected wheel as noise and will remove it completely. Also, if a shape shares the contour with some noise elements, the final shape will be distorted, as all connected noise will not be excluded. We also observed some images in the dataset which were handwritten names of the elements instead of doodles(e.g. Handwritten word mug instead of a drawing of a mug).

6.2 SVM

The baseline approach, SVM with linear kernel, performed as expected. Image classification task is a tough task for this algorithm because of a large amount of overlap between the classes. Especially since we trained the model on individual pixel intensity values, the overlap can be very significant, which leaves no clear place for the separating hyperplane to be placed.

Thus, it would be interesting to try to train a SVM classifier on pre-processed images, which contain more information about the shapes and textures, rather than the individual intensities of the pixels. This way we can reduce the number of features and possibly the amount of overlap between images, as we later see with CNN.

6.3 Neural Net

While more computationally intensive, the Neural Network is able to find non-linear solutions and, unlike SVM, distinguish between overlapping classes. This is especially helpful for image classifications, since images tend to have classes that are similar and only have a small feature that distinguishes them, for example distinguishing between a drawing of a nail and one of a pencil.

Using a cross-validation set for the Neural Network increased the training time significantly, but ensured that our validation performance metric was reliable. Updating the weights after each sample instead of after each epoch ensured that no local minimums were mistaken for the true minimum.

When choosing the best number of layers, we used the same number of nodes in each hidden layer. In the future, with more time and access to better machines, it would be interesting to try a wide variety of different numbers of nodes at each layer to better fit the model. In this case, a more highly layered model could result in better accuracy.

The main issue with NN is that it is very heavy computationally, yet still processes each pixel as a separate feature, which is less useful when we are trying to recognize an entity in which the values of the neighboring elements are also important.

6.4 Convolutional Neural Net

Unlike our simple NN, CNN takes a 2-d array (tensor) as input and can therefore take into account the relative position of each pixel in the picture. This is an advantage, as the feature extraction steps of the CNN really do extract separate shapes and their approximate positions on the image.

The point that really does differentiate the CNN from a conventional NN, is the automation of feature extraction. One could theoretically come up with good filters manually and pre-process the images before feeding them into the NN, but CNN can extract it automatically and picks the best filters without human supervision. Furthermore, it would be extremely difficult for a human to design a large number of filters and then to design useful filters for the previously filtered image. In addition, the CNN framework allowed us to augment the training dataset, which significantly improved our performance by training a model that was robust to slight variations of the image. We resorted to modifications of small magnitude, so in future it would be interesting to try higher values for allowed modifications.

On the more practical side, a finely tuned feature extraction method reduces the complexity of the learning task by simply reducing the number of input and necessary nodes and layers. This provides better computational efficiency and better results in the end.

For future work, it would be proper to use cross-validation for model selection, since, due to a wide range of CNN hyperparameters, limited time and computational resources, we had to resort to using a single validation set.

7 Statements of Contribution

1. Defining the problem $\rightarrow$ All team members contributed during the meetings.
2. Developing the methodology $\rightarrow$ Mostly completed by Gael Duong and Richard Hughes
3. Coding the solution $\rightarrow$ Each of the members did their respective parts, Gael Duong had by far the largest part, working on pre-processing, first SVM baseline and the CNN, Richard Hughes was tasked with coding the NN from scratch and Dmitrii Tiron did the final extended

evaluation of the SVM

4. Performing data Analysis $\rightarrow$ All members contributed equally
5. Writing report $\rightarrow$ Mostly completed by Dmitrii Tiron, with heavy contributions from Gael Duong and Richard Hughes

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