STA 141C Final Project Report

Evan Dumas & Sandra Bae

Overall Experimental Goal

In this project, we used sci-kit learn and Keras to compare different machine learning algorithms (i.e. K-Nearest Neighbors, Random Forest, and Convolutional Neural Network) on classifying doodles from Google's *Quick, Draw!* game. In this game, people are told what they should draw in less than 20 seconds, while a neural network is predicting in real-time what it sees in the drawing. Once the prediction is correct, you can't finish your drawing, which might explain the rather minimalistic style of the drawings. Run-time and accuracy will both be used to rank each of the algorithms and determine the best algorithm overall.

Discussion of Dataset

Our dataset is a subset of the hand-drawn doodles from Google's *Quick, Draw!* dataset which is composed of 50 million drawings of 345 different categories. For the sake of our experiment, and for feasibility reasons, we chose to use a subset of the overall dataset and focus on the "sheep", "cat", "book", and "basket" categories by using 5000 images from each to generate an overall dataset of 20,000 images, and of which each has been preprocessed to a uniform 28x28 pixel image size. This larger 20,000 image dataset was then randomized and split evenly into a training and testing set (this shall be referred to as the full dataset). Additionally, to test the effect of doubling the number of categories on our classification algorithms, we generated a smaller dataset composed of only the "cat" and "sheep" images which contained a total of 10,000 images (5000 from each category) and was also randomized and evenly split into a training and testing set (this dataset will henceforth be referred to as the binary dataset).

Each entry of the dataset was in the form of a 28x28 gray-scale numpy bitmap that had been reshaped into an array of 784 elements. Each of the image categories were labelled as follows: "cat" => 0, "sheep" => 1, "book" => 2, and "basket" => 3.

Algorithm Implementations/Justification:

Because the performance of classifiers is inherently data dependent, we have decided to try three different algorithms in order to help us figure out which is the best classifier for Google's QuickDraw dataset.

Random Forest:
For the implementing the Random Forest Algorithm, we used the 'RandomForestClassifier' from 'sklearn.ensemble' on both the full and binary dataset. For the binary dataset, we have tested the accuracy of the algorithm with different n_estimators (i.e. the number of trees in the forest). Once we have found that n=100 was the most optimal parameter, we have used this parameter set forth for the multi-class classification. However, first, we tried out a simple random forest classifier using the default options

(except for random_state for reproducibility and n_jobs=-1 to use multiple CPUs) from 'sklearn.ensemble'. The reason we chose Random Forest was because we knew the end goal was having multi-class classifications. In addition, because Random Forest works well with a mixture of numerical and categorical features we felt it would be interesting to use Random Forest with the data in its current state (.npy).

K-Nearest Neighbors:
For implementing the K-Nearest Neighbors algorithm, we used the `KNeighborsClassifier()` from `sklearn.neighbors` on both the full and binary dataset while varying the `n_neighbors` parameter to range from 1 to 5. For each value of `n_neighbors`, the model was fitted to the training dataset using `KNeighborsClassifier.fit()` and the accuracy was determined using `KNeighborsClassifier.score()`. The runtime of the algorithm was determined using the `time()` function in the python `time` library.
We chose the K-Nearest Neighbors algorithm as one of our algorithms for classifying image data because it is the simplest and easiest machine-learning algorithm for classification. It makes sense to initially start with this algorithm due to its simplicity in implementation and method for classification. Since most of the image categories should have very different pixel structures from each other (only cat and sheep should look similar as they are structurally similar objects), it would make sense that a K-Nearest Neighbors approach would have high accuracy in classifying the images in our dataset.

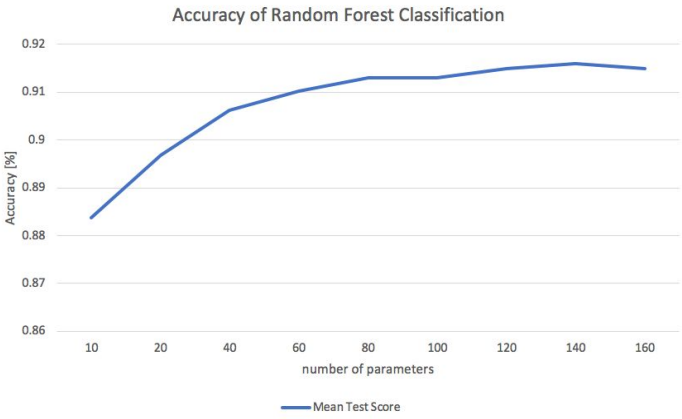
Convolutional Neural Network:
Let's try out a Convolutional Neural Network (CNN) with Keras. We will use a model from this tutorial by Jason Brownlee. It has the following 9 layers:

1. Convolutional layer with 30 feature maps of size 5×5.
2. Pooling layer taking the max over 2*2 patches.
3. Convolutional layer with 15 feature maps of size 3×3.
4. Pooling layer taking the max over 2*2 patches.
5. Dropout layer with a probability of 20%.
6. Flatten layer.
7. Fully connected layer with 128 neurons and rectifier activation.
8. Fully connected layer with 50 neurons and rectifier activation.
9. Output layer.

In contrast, Convolutional Neural Networks (CNN) are proven to be effective in areas such as image recognition and classification. However, compared to Random Forest, we would need to reshape the data into the necessary format for the neural network to work with. But because this task is inherently image classification, we wanted to include CNN to see how other algorithms would compare to it.

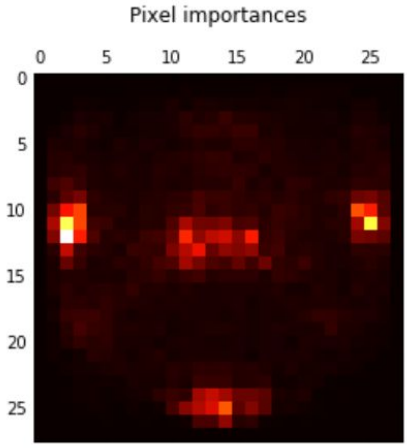
Results

(1) Random Forest



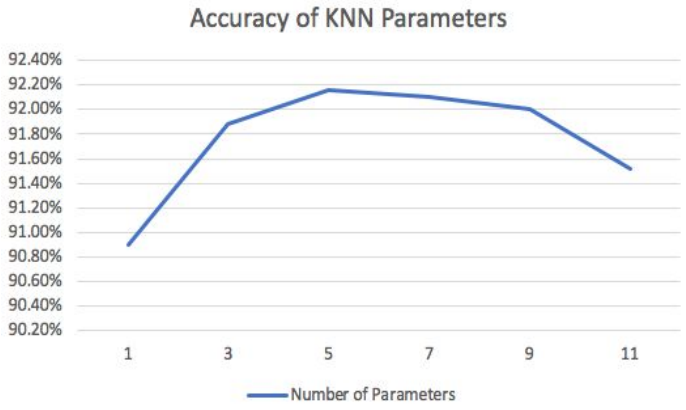
We wanted to first the find the most important argument of the algorithm: n_estimators, which tells us the number of trees in the forest. Though it is set to 10 by default, we wanted to experiment by trying larger values as we believe this should increase the accuracy. This is due to the fact as there is a proof of convergence of the random forest accuracy when the number of trees goes towards infinity. If our implementation of the random forest is correct, we should see an increase of accuracy.

And from our results, we can see that this is the case. Please refer to our appendix to see the full table. But from the default parameter (n = 10) had an accuracy of 88.38% from the binary classification and it continued to increase so until around 100 trees. After 100 trees, we see an accuracy plateau which fits accordingly to the convergence proof.



Using Random Forest, we can also visualize the model to see which pixels are the most important by using sklearn's 'feature_importances_' property. We reshaped it back to a 28x28 pixel and put a color map over the plot. And as expected, corners have a very low importance, as they are mostly blank. The left, right, middle and bottom part of the picture are most important. Looking at the pictures of cats and sheeps, it seems that the middle is mostly blank in sheeps, but not in cats. So pixels there probably indicate a cat image. The other 'hot' areas are less clear to me after a quick inspection of the images

(2) K-Nearest Neighbor:

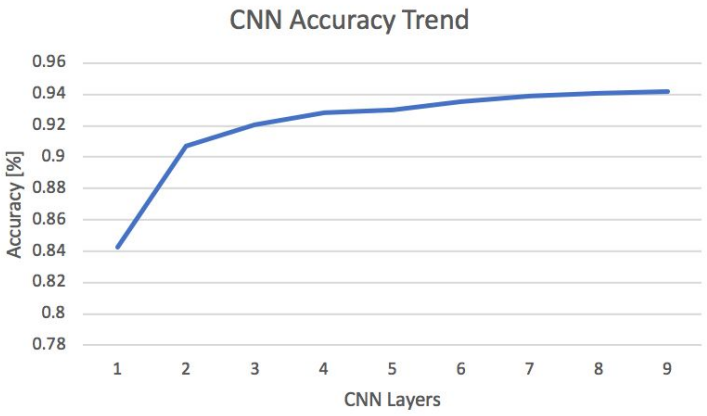


The optimal k-Neighbors parameter choice was voting on the classification using 5 of the nearest neighbors. For the k-Neighbors values ranging from 1 to 4, there was a positive response in the accuracy from increasing the value of k-Neighbors. The maximum accuracy was achieved at a k-Neighbors value of 5 followed by a decrease in the accuracy for

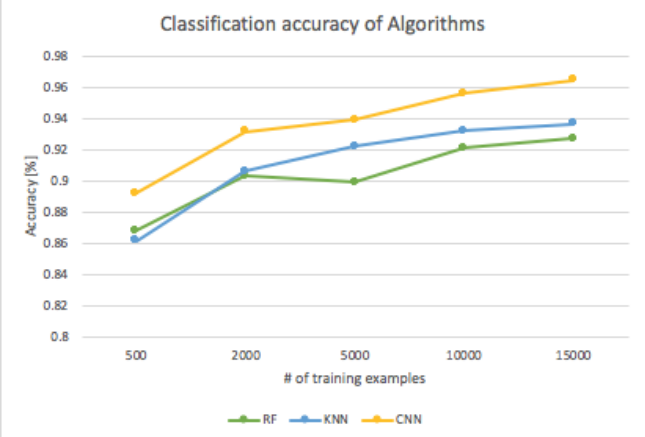
k-Neighbors equal to 6. It also worth noting that k = 5 is the default option. In addition, this result follows the trend that the optimal value for k-Neighbors in an odd number since our data contains an even number of categories (binary) which can lead to ties when voting for classification. The decrease in accuracy for k-Neighbors values above 5 is most likely due a loss of distinction between the boundaries of the classes. In the end, the KNN classifier is quite a bit more accurate than the random forest, but it also takes much longer (particularly the classification of new examples).

(3) CNN

As mentioned in the earlier part of our report, we are using a model by Brownlee. Because this particular model happened to have 9 layers, we were curious if there would be underfitting (high bias), overfitting (high variance), or it would be a good compromise. As it would be both of our first time to use a deep learning technique, we first wanted to see the accuracy trend for this model. For example, in the case of classifying between apples and grapes, we would want to declare the distinguish the possibly features on the basis of visual features (i.e. curves, shape, color). But adding unnecessary layers will not only increase the parameters, but also accuracy.



But in our case, we deduced that it is a good compromise of variance and bias as we can see a steady increase in terms of accuracy. In fact, compared to the two previous algorithms, CNN, not surprisingly, had the highest accuracy for the binary classification. However, the biggest drawback was it required the longest training time.



Lastly, we wanted to compare the classification accuracy of the three algorithms over different training examples used.

From the figure, we can see that CNN is the leading contender as its accuracy continues to increase over the the number of training examples. However, we wanted to extend this further to multi-class classification and see if it would be make any difference to its accuracy and runtime performance.

	Binary Classification		Multi-class Classification	
	Accuracy [%]	Wall Time Performance	Accuracy [%]	Wall Time Performance
RF	91.60%	27.7s	79.10%	8.7s
KNN	92.18%	2min 50s	82.36%	1min 40s
CNN	94.16%	4min 10s	90.28%	10min 53s

Conclusion

Based on the accuracies and runtimes, we chose the CNN as the optimal machine-learning algorithm for classifying our dataset. The CNN is most likely the best algorithm for classifying image data because it is able to break up the images into multiple features/layers and use methods (like gradient descent) to determine the appropriate weight for each feature. These weights are then further adjusted via back-propagation thus making CNNs the most effective at classifying image data (which are composed of features like edges and colors). However, as mentioned, the CNN required the greatest amount of time for training due to the feature training and back-propagation steps thus there are runtime drawbacks that come from using a more accurate algorithm.

Difficulties

Since we were working with image data and utilizing open source packages that were not covered in class, we had several difficulties while completing the project. Our first and most critical difficulty was determining how to construct our dataset from the numpy bitmap files on the Google Cloud server. Most of the open-source software packages expect to receive a shuffled dataset with an x-column containing the image arrays and a y-column with a numerical label for each category in the dataset. Using numpy functions we were able to restructure the individual numpy bitmap files into one large dataset containing an equal number of samples from each category and a numerical label for each sample. Another difficulty we faced was setting up the Convolutional Neural Network since it required us to learn how to use `keras`, an open-source package neither of us had used before. After watching several online tutorials and reading up on `keras` and Convolutional Neural Network implementations we were able to construct a functional neural network.

Appendix

Random Forest Accuracy:

Number of Parameters	Accuracy
10	0.8838
20	0.8968
40	0.9062
60	0.9102
80	0.9102
100	**0.9130**
120	0.9148
140	0.9150
160	0.9160

KNN Accuracy:

Number of Parameters	Accuracy
1	90.90%
3	91.88%
5	**92.16%**
7	92.10%
9	92.00%
11	91.52%

CNN Accuracy:

Epoch	Accuracy
1	0.8426
2	0.9072
3	0.9208
4	0.9284
5	0.9302
6	0.9354

7	0.9392
8	0.9408
9	**0.9416**

Comparison of Algorithms (Different # of Examples Used)

	Number of Training Samples				
	500	2000	5000	10000	15000
RF	0.868	0.9030	0.8992	0.9210	0.926867
KNN	0.862	0.9060	0.9218	0.9322	0.936400
CNN	0.892	0.93.15	0.9392	0.9560	0.964200