Doodle Classification Report

Computer Vision is a field of Artificial Intelligence and Computer Science that deals with how computers can be made to gain an understanding of digital images and videos. With the advances in machine learning technology, innovations in Image Processing are emerging rapidly and being leveraged by businesses all around the world. Applications of Computer Vision include:

1. Boredom killers such as your favorite Snapchat filter that use facial recognition and image processing to distort your face and/or overlay images
2. Potential lifesaving uses such as Medical startups that claim they'll soon be able to use computers to read X-rays, MRIs, and CT scans more rapidly and accurately than radiologists.
3. And borderline unethical uses such as Facial Recognition in police body cams

With so many emerging and interesting uses of Computer Vision, my goal is to compare different machine learning algorithms from scikit-learn and Keras tasked with classifying drawings made from Google's Quick Draw! dataset. By doing so I will showcase my abilities processing and analyzing the doodles through image classification and neural networks. This project is meant for companies that are interested in leveraging the Computer Vision methods used in my project in business.

Data

Google has capitalized on the use of crowdsourcing to label over 50 million drawings with their online game "Quick Draw!". It gives its users 20 seconds to draw one of 345 different classes that range from an aircraft carrier to a zig-zag. Recently, they have open-sourced all their data and I will select 10 images to build my classifier. The data we used is a 28x28 grayscale bitmap in numpy .npy format of the simplified version of each drawing.

Other datasets we could have used include:

- The *raw dataset* that contains a two letter country code of where the drawing originated, the drawing in a JSON array representing the vector drawing that details each stroke in the drawing and the timing it took to complete each stroke, whether the word was recognized by the game, the label, and the date/time of when the drawing was created.

- A *simplified version* of the raw dataset that has timing information removed and has the drawings positioned and scaled into a 256x256 region.

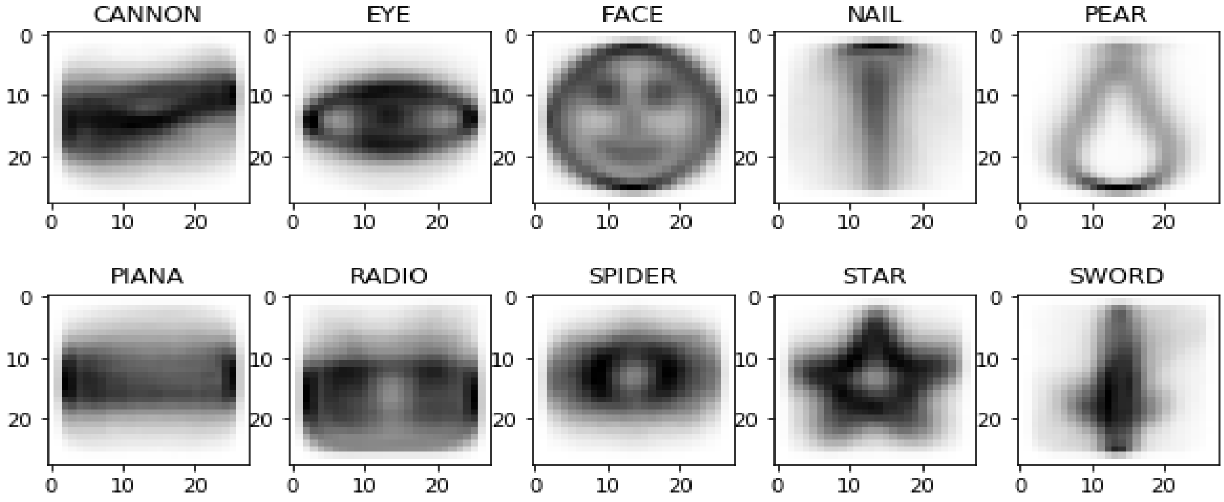
The **data wrangling** process consisted of:

- Loading each class separately into their own numpy array
- Adding the class label to the array of each sample in the dataset.
- The samples were then aggregated and split into training and testing sets.
- The features were normalized between 0 and 1, as a means to "center" the data and have all pixels on the same scale. This weights all features equally in their representation.

Initial Findings

We started our exploratory analysis by taking the first image in each classes' dataset and plotting them along side a histogram of the pixel intensities. The histogram showed that each doodle's pixels were mainly grouped at the two extreme of intensities with very few falling between. Most pixels in the dataset are completely white, along with another set of pixels that are completely dark, with relatively few in between. If we wanted to, we could probably replace each pixel with a binary 0 or 1 with very little loss of information.

From there we plotted the average doodle for each class and the average doodle of all classes together. The average doodle for each class really highlighted where the important pixel will be for each class.



Classification Modeling

By simply randomly guessing, one should be able to reach ~10% accuracy (since there are only ten class labels). A machine learning algorithm will need to obtain > 10% accuracy to demonstrate that it has in fact “learned” something (or found an underlying pattern in the data).

For all the models, except the Convolutional Neural Network and the Transfer Learning model (VGG16), we used GridSearchCV to tune the most important hyper-parameters.

K-Nearest Neighbors

To start, I modeled the data with the k-Nearest Neighbor (k-NN) classifier, which is arguably the most simple, easy to understand machine learning algorithm. The k-NN algorithm classifies unknown data points by finding the most common class among the k-closest examples. Each data point in the k closest examples casts a vote and the category with the most votes is chosen. This model will serve as a good benchmark to compare the rest.

Using GridSearchCV and the elbow-method to tune the *k-neighbors* parameter we found that 3 neighbors seems like the best choice to avoid overfitting. The main advantage of the KNN algorithm is that it performs well with multi-modal classes because the basis of its decision is based on a small neighborhood of similar objects. Therefore, its results were fairly high with 80%. The main disadvantage is the computational cost are very high and the results take far too long. Also, this classifier would not be ideal if we wanted to make predictions on the fly, since k-NN classifier trains the data fast but then makes predictions very slow.

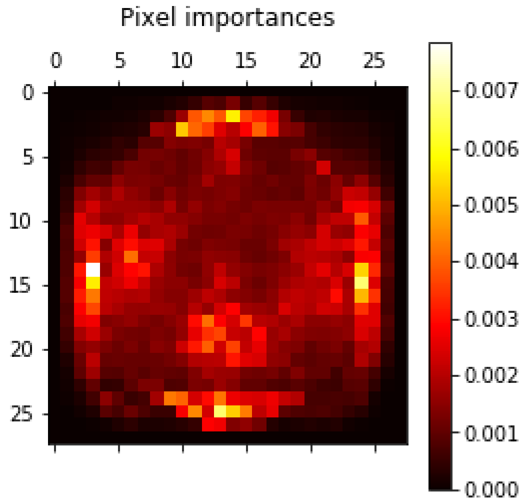
Our result for KNN was **0.8037**. This is a good baseline to compare the other models.

Random Forest

Random forests is an ensemble model which means that it uses the results from many different models to calculate a label. For our model we tuned *n_estimators* which is the number of trees you built before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower. Using the elbow method again, show us **80 estimators** was a good choice since the number of estimators above 80 did not improve the model enough to be worth the extra computational requirements.

We then tuned the *max_features* parameters, which are the maximum number of features Random Forest can try in individual tree. By limiting the max features to the **square root of total features** we improved the model and made it computationally less expensive.

We then plotted the pixel importances and saw that the edges of the doodles tend to be the most important.



Our results for the Random Forest Classifier was **0.8048.** The random forest had an accuracy score very close to the kNN model.

Support Vector Machine

SVM classification uses planes in space to divide data points. We have created two models with different types of dividing planes (Linear and Non-Linear)

The linear model (LinearSVC) had an accuracy of **0.7178**, while the non-linear model (SVC with Radial Basis Function) had an accuracy of **0.7815**. While the non-linear model was better, both did not outperform either the k-NN model or the Random Forest model.

Multi-Layer Perceptron

A perceptron is a neural network with a very basic architecture. This will be good to compare against the convolutional neural network. Neural Networks receive an input (a single vector) and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.

For tuning the hyper-parameters for a Multi-Layer Perceptron, we tried different number of layers (1-3 hidden layers) and number of neurons in each layer (100 vs 758). The default activation function (ReLU) is the most effective choice from sklearn and the default optimization algorithm (Adam) is the best choice due to the size of our dataset. The best model was the one-layer model with 758 neurons.

The Multi-Layer Perceptron has an accuracy of **0.8299**, which so far is the best model.

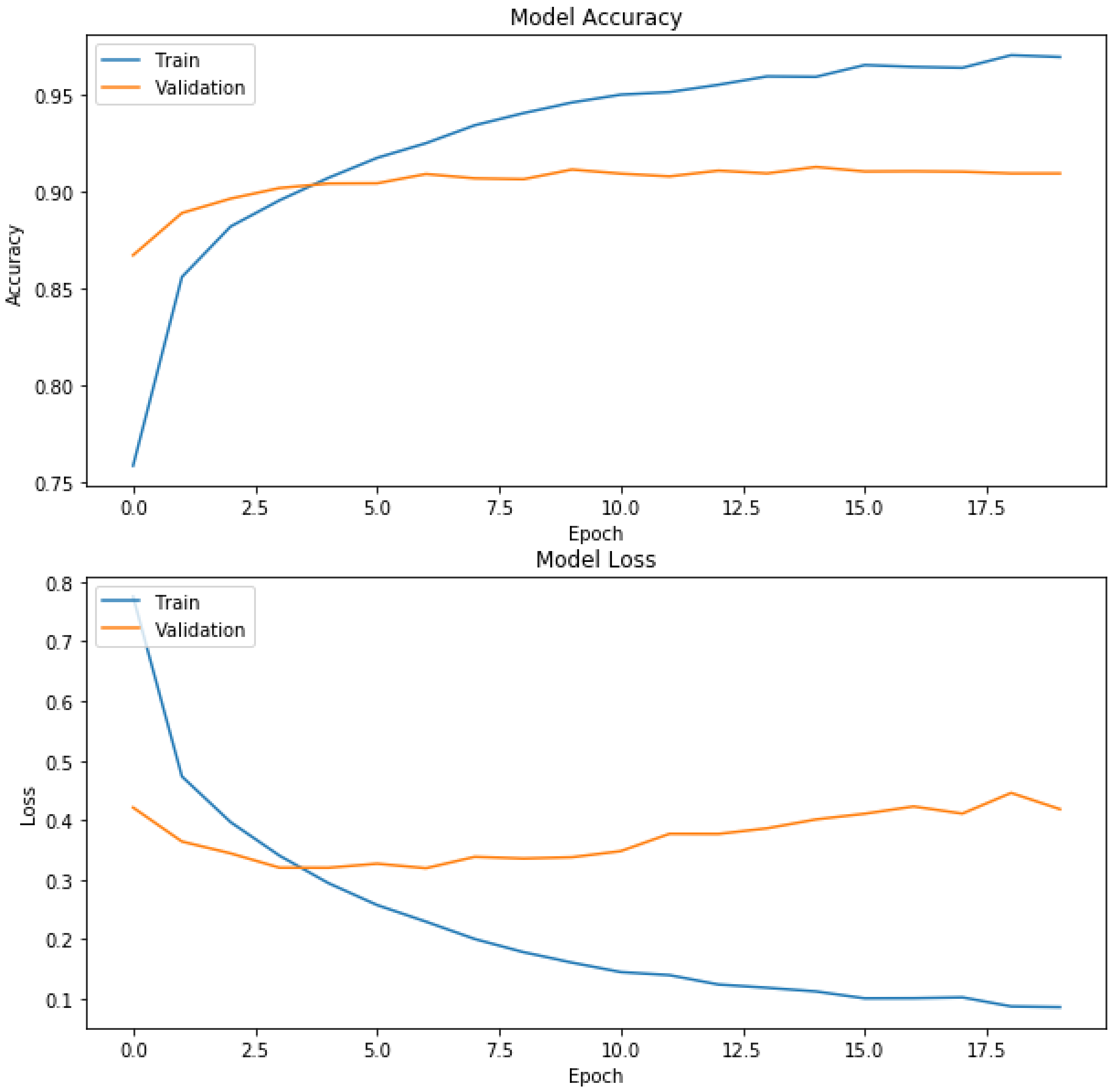
Convolutional Neural Networks (CNN / ConvNet)

The Convolutional Neural Network architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network.

We reshaped the input for the CNN into a 4-dimensional array with the shape of (# of doodles, width, height, and depth(color channel)) so that each image is a 3-dimensional image. Our final shape was (21000,28,28,1) because the color channel was greyscale. A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We used a commonly used architecture of INPUT -> (CONV -> RELU -> CONV -> RELU -> POOL) -> (DO ->FC -> RELU) -> (DO -> FC).

- The first CONV layer we used a filter with 5 pixels width and height and then used with the ReLu activation function. We used the ReLu activation function because it is very simple and efficient, and it avoids and rectifies vanishing gradient problem . Almost all deep learning Models use ReLu nowadays. The second CONV layer used 3 pixels width and height instead.

- The two dropout layers are less common but were used as a regularization. When random neurons are dropped out, the network is forced to learn several independent representations of the patterns with identical input and output. The generalizability thus improves.
- We then compiled the model using 20 epochs are the resulting training accuracy and loss and validation accuracy and loss can be seen in the image below.

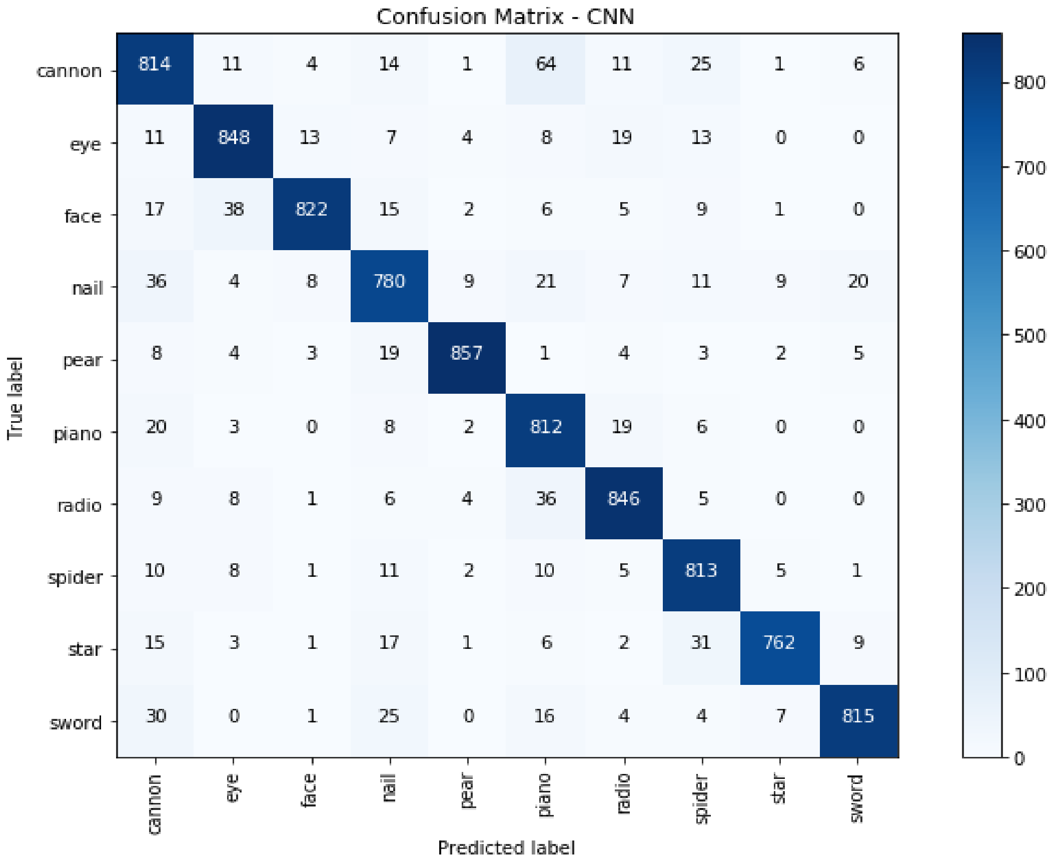


While it is possible to try many different variations of a CNN model, our results were impressive enough that any improvements would not be worth the computational effort. Our final accuracy was **0.9095**, which shows how powerful deep learning is.

```
             precision    recall  f1-score   support

     cannon       0.84      0.86      0.85       951
        eye       0.91      0.92      0.92       923
       face       0.96      0.90      0.93       915
       nail       0.86      0.86      0.86       905
       pear       0.97      0.95      0.96       906
      piano       0.83      0.93      0.88       870
      radio       0.92      0.92      0.92       915
     spider       0.88      0.94      0.91       866
       star       0.97      0.90      0.93       847
      sword       0.95      0.90      0.93       902

avg / total       0.91      0.91      0.91      9000
```



Confusion Matrix - CNN

	cannon	eye	face	nail	pear	piano	radio	spider	star	sword
cannon	814	11	4	14	1	64	11	25	1	6
eye	11	848	13	7	4	8	19	13	0	0
face	17	38	822	15	2	6	5	9	1	0
nail	36	4	8	780	9	21	7	11	9	20
pear	8	4	3	19	857	1	4	3	2	5
piano	20	3	0	8	2	812	19	6	0	0
radio	9	8	1	6	4	36	846	5	0	0
spider	10	8	1	11	2	10	5	813	5	1
star	15	3	1	17	1	6	2	31	762	9
sword	30	0	1	25	0	16	4	4	7	815

Cannon was the class with the lowest f1-score and also had the most labels in the testing set. The pear had the highest f1-score which makes sense, because it was one of the simplest drawing. The piano is being predicted . The piano had high recall which means it was was predicted well for itself, but the model predicted piano too often for other labels making its precision low. The piano was mainly predicted inaccurately when the true label was a cannon, a radio, or a nail. All these shapes share the base shape of a rectangle, so it was probably doodles with little details that confused the model. The labels that had more circular doodles (pear, eye, face) had higher f1 scores.

Transfer Learning - VGG16

Convolutional Networks are rarely built from scratch due to the amount of computational power it takes or not having enough samples per class. Transfer learning provides the ability to use pre-trained models learning as the inputs to your newly created model. Transfer learning is the ability to store knowledge gained while solving one problem and applying it to a different but related problem. The VGG neural network is an image classification convolutional neural network that was trained on over 1.2 Million images from the ImageNet set and spans over 1000 classes. To get the VGG16 to work on our dataset, we had to resize the height and width to a minimum of 48 and have a 3-color channel. We changed the input shape for each image to (56,56,3).

Our resulting accuracy was **0.9213** which is even higher than our CNN model that was made from scratch. This just show how powerful and simple it could be to use pre-trained neural networks.