Classification Models for Sketch Drawings

Aparajita Dutta
Department of Electrical Engineering
Stanford University
apdutta@stanford.edu

Sarah Saki Robinson
Department of Computer Science
Stanford University
srobins2@stanford.edu

Abstract

In recent years, there has been significant development in the field of computer vision. Numerous papers have been published, analyzing various algorithms for image classification on photographs. In this paper, we apply different classification techniques to a dataset of sketch drawings to find what model performs best.

1 Introduction

A large percentage of publications on image classification perform experiments on photographs where the objects in images are easily identifiable by the human eye. In this paper, we explore a scenario where the images of objects are not as easily identifiable by humans. We compare different classification models applied to a dataset of doodles and analyze which model performs best on the drawing data. The models applied in this paper span from simple machine learning classifiers like Logistic Regression and Naive Bayes, to deep learning methods such as Convolutional Neural Networks and Multi-Layer Perceptron.

2 Related work

There is a lot of literature available in the area of image classification. Tianmei G. et al. propose a simple Convolutional Neural Network model for image classification in their paper [1]. For our project we plan to implement a similar model and compare its performance on sketch data with other machine learning models.

In their paper [2], David H. et al. use the Quickdraw dataset to generate new line drawings using a recurrent neural network model. Although the objective of this paper is different from our classification project, the paper contains of lot of valuable information about the nature of the dataset we will use for this project.

3 Dataset and Features

For this project , we used the Google QuickDraw dataset [3]. The dataset contains a large number of simplified drawings, centered and formatted to a 28×28 gray-scale bitmap. Although the data was available for over 300 different objects, only a small subset containing 22 objects was chosen for this project, namely: Apple, Banana, Bicycle, Birthday Cake, Butterfly, Candle, Computer, Door, Drums, Firetruck, Hat, Horse, Ice Cream, Leaf, Panda, Peanut, Pencil, Rainbow, Smiley Face, Snowman, Soccer Ball and Umbrella. Figure 1 shows a sample of the data used for this project.

We restricted the data to N number of images per class, where N is a hyper-parameter that was tuned later in the experimentation. This was done to reduce computation time and induce class balance. The final, formatted data was then split in a 60:20:20 ratio into the training, development and test sets respectively.

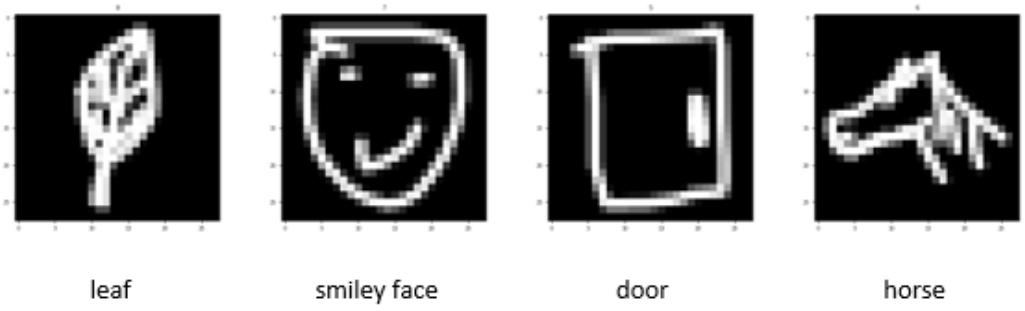
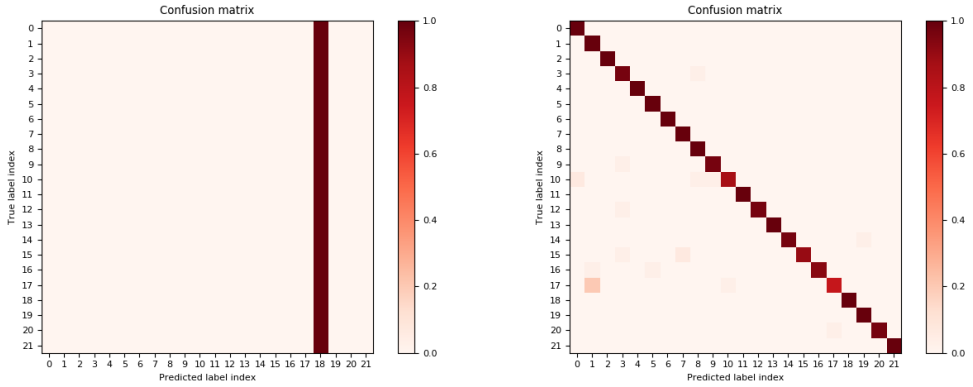


Figure 1: Sample images from the dataset along with the true labels

4 Baseline and Oracle

To create the baseline expectation for this project, we loaded the data with $N = 20000$ images per class, and implemented a majority classifier. This achieved 4.447% accuracy with log loss = 33.003 on the development set.



(a) Baseline

(b) Oracle

Figure 2: Confusion Matrices for Baseline and Oracle

For the oracle, we simulated a real-human classifier. We did this by printing out 30 individual images for each of the 22 classes and hiding the true labels. We then shuffled the images to randomize the order and attempted to identify the correct class for each image. With this model, a 96.818% accuracy was achieved with log loss = 1.099. Figure 2 illustrates the confusion matrices for the baseline and oracle respectively.

5 Methods

5.1 Multi-Class Logistic Regression

Multi-Class Logistic Regression or SoftMax Regression is a generalization of Logistic Regression where the sigmoid logistic function is replaced by the SoftMax function, ϕ, to allow multi-class classification. This function is obtained as:

$$P(y = i|x; \theta) = \phi_i = \frac{e^{\theta_i^T x}}{\sum_{j=i}^{k} e^{\theta_j^T x}}$$

where k is the number of classes. In this classification problem, $k = 22$ and the model parameters are $\theta_1, ..., \theta_{22}$.

Logistic regression is a quick and simple classifier. In this project we want to observe how well it performs on the drawing dataset in comparison to more complex and slower deep learning methods.

5.2 Naive Bayes

Naive Bayes is a supervised learning algorithm that uses Bayes' theorem to get the relationship:

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)}$$

The classifier is based on the strong assumption that the features, $x_1, ..., x_n$, are conditionally independent given the labels, y.

Although this algorithm is more commonly used in text classification, we are interested in observing how well is performs on images. There are different variations of Naive Bayes where different assumptions are made about the distribution of feature likelihoods. We will be implementing and comparing 3 different forms on Naive Bayes in our experimentation where the feature likelihoods are assumed to be distributed according to the Gaussian, Multinomial or Multivariate Bernoulli distributions.

5.3 Multi-Layer Perceptron (MLP)

MLP is a simple, feed-forward neural network model that uses layers with different non-linear activation functions to perform supervised learning. The model consists of an input layer, one or more hidden layers and an output layer.

For this project we create a simple MLP model described as follows: the first layer is a flatten layer since MLP uses simple vectors. Next is a fully connected layer with 1000 neurons, that uses the rectified linear unit (ReLu) activation function: $f(x) = max(0, x)$. Then there is a dropout layer which prevents over-fitting by dropping some neurons during each training epoch. This pair of a fully connected layer with dropout is repeated once more with 512 neurons and finally we add a fully connected layer with 22 (number of classes) neurons and a SoftMax activation function for the output layer. Figure 6 in the Appendix shows the model plot.

5.4 Convolutional Neural Network (CNN)

CNNs are a regularized version of MLP which trains on the image data while still keeping the dimensions of the image intact. A convolutional layer operates on an image piece by piece by making use of pooling layers.

For this project, we experimented with 2 different network architectures. The first model, CNN_1, consists of 3 convolutional layers, with 16, 32 and 64 filters respectively, each with a 2×2 max pooling layer with stride 2. This is followed by a dropout layer, a flatten layer and 2 fully connected layers with 500 and 22 neurons respectively. Each non-output layer uses the ReLU function for activation while the output layer uses the SoftMax function. Figure 5a in the Appendix illustrates this architecture.

The second CNN model used for this project, CNN_2, uses more fully connected layers than convolutional layers. The models starts with 2 convolutional layers with 32 and 64 filters respectively, each with a 2×2 max pooling layer with stride 2. Next there is a flatten layers followed by 3 fully connected layers with 512, 128 and 22 neurons respectively. This time we use a dropout layer after each non-output, fully connected layer. Once again, all the non-output layers use ReLU activation and the output layer uses SoftMax activation. Figure 5b shows the plot for this architecture.

6 Experiments and Results

We started off the experimentation by choosing a value for the hyper-parameter N, described in Section 3. This was done by testing different values of N and applying a simple logistic regression model to the data. The results from this experiment can be found in Table 1. We observed that for small Ns, the model seemed to be over-fitting on the training set, as the training log loss was significantly lesser than the development log loss. Even though, $N = 50000$ obtained the best results,

the computation time was very high. Therefore, $N = 20000$ was chosen for further experiments as the results were comparable enough to $N = 50000$, while still maintaining a manageable computation time.

In the following subsections, we describe how we tuned the different models individually and report the results obtained in each case. The reported evaluation metrics are the the log loss, accuracy score and macro-averaged precision and recall obtained on the development set, the log loss obtained on the training set, and the confusion matrix obtained for the best set of hyper-parameters for each method.

N	Train Log Loss	Log Loss	Accuracy	Precision	Recall
1000	7.7738	12.1121	0.6493	0.65	0.65
5000	9.4971	10.6771	0.6908	0.69	0.69
10000	9.7462	10.5107	0.6957	0.69	0.70
20000	9.9290	10.2584	0.7030	0.70	0.70
50000	9.9360	10.0669	0.7085	0.71	0.71

Table 1: Tuning the number of images per class (N)

6.1 Multi-Class Logistic Regression

Logistic Regression was implemented using stochastic gradient descent with L2-regularization. To create the model in Python, we used the Scikit-learn library [4]. The inverse of the regularization strength, C, was taken as a hyper-parameter that was tuned using grid search. Table 2 shows the results for different values of C.

As seen from the results, setting $C = 0.1$ achieved the best accuracy and log loss. Figure 3a shows the confusion matrix obtained with $C = 0.1$.

C	Train Log Loss	Log Loss	Accuracy	Precision	Recall
0.01	10.1267	10.2953	0.7019	0.70	0.70
0.1	9.9290	10.2584	0.7030	0.70	0.70
0.5	9.8735	10.2733	0.7026	0.70	0.70
1	9.8654	10.2729	0.7026	0.70	0.70

Table 2: Tuning regularization strength inverse (C) for Logistic Regression

6.2 Naive Bayes

As mentioned in Section 5.2, three variations of Naive Bayes were implemented for the experimentation and the results were compared to determine which distribution assumption for the feature likelihood works best for our drawing data. The Scikit-learn library [4] was used to create the Naive Bayes models. Table 3 shows the results obtained in each case.

Bernoulli Naive Bayes achieved the best results on the development set. Figure 3b shows the confusion matrix obtained with Bernoulli Naive Bayes.

Model	Train Log Loss	Log Loss	Accuracy	Precision	Recall
Gaussian	18.8616	18.8476	0.4543	0.51	0.45
Multinomial	16.8599	16.8369	0.5125	0.52	0.51
Bernoulli	15.6298	15.5872	0.5487	0.56	0.55

Table 3: Results using different Naive Bayes models

6.3 Multi-Layer Perceptron

For the implementation, we used the Sequential model from Python's Keras Library [5] that allowed us to create the simple neural network described in Section 5.3. To find the best combination of batch

size, b_{MLP}, and the number of epochs, e_{MLP}, we ran a series of experiments. The results for these experiments are shown in Table 4. As seen from the table, $b_{MLP} = 100$ and $e_{MLP} = 3$ achieved the lowest loss with least over-fitting. Figure 3c shows the confusion matrix obtained with the tuned model.

b_{MLP}	e_{MLP}	Train Log Loss	Log Loss	Accuracy	Precision	Recall
10	1	10.2427	10.3966	0.6990	0.70	0.70
10	3	13.4922	13.6852	0.6038	0.79	0.60
10	10	14.4456	14.5437	0.5789	0.77	0.58
10	50	18.2932	18.2977	0.4702	0.71	0.47
50	1	5.8011	6.2060	0.8203	0.83	0.82
50	3	5.8999	6.4564	0.8131	0.83	0.81
50	10	7.7617	8.4110	0.7565	0.81	0.76
50	50	16.3765	16.5076	0.5221	0.70	0.52
100	1	4.8369	5.3374	0.8456	0.85	0.85
100	3	4.5148	5.2632	0.8476	0.86	0.85
100	10	4.2221	5.2915	0.8468	0.85	0.85
100	50	5.6027	7.0840	0.7949	0.82	0.79

Table 4: Results using different batch size and epochs for MLP

6.4 Convolutional Neural Network

Similar to the MLP model, the two CNN models described in Section 5.4 were implemented using Python's Keras Library [5]. We ran individual experiments for the batch size and number of epochs for both the models and compared the final results to find the best combination of architecture, batch size and epochs. The results for the experiments with the CNN_1 model, Figure 5a, are shown in Table 5 and the CNN_2 model, Figure 5b, are shown in Table 6. As seen from the results, the second model performed slightly better than the first and $b_{CNN_2} = 100$ and $e_{CNN_2} = 10$ achieved the lowest loss. Figure 3d shows the confusion matrix obtained with the best model.

b_{CNN_1}	e_{CNN_1}	Train Log Loss	Log Loss	Accuracy	Precision	Recall
10	1	4.9530	4.9551	0.7911	0.86	0.86
10	3	6.5474	6.5815	0.8094	0.82	0.81
10	10	13.1444	13.0325	0.6194	0.67	0.62
10	50	18.1778	18.1375	0.4748	0.53	0.47
50	1	32.5558	32.5681	0.0570	0.04	0.06
50	3	3.1423	3.2376	0.9062	0.91	0.91
50	10	3.5186	3.6418	0.8945	0.90	0.89
50	50	9.6347	9.6190	0.7215	0.77	0.72
100	1	4.1781	4.1627	0.8794	0.88	0.88
100	3	2.8324	2.9687	0.9140	0.91	0.91
100	10	2.4777	2.6630	0.9228	0.92	0.92
100	50	3.0696	3.1834	0.9078	0.91	0.91

Table 5: Results using different batch size and epochs for CNN_1 architecture

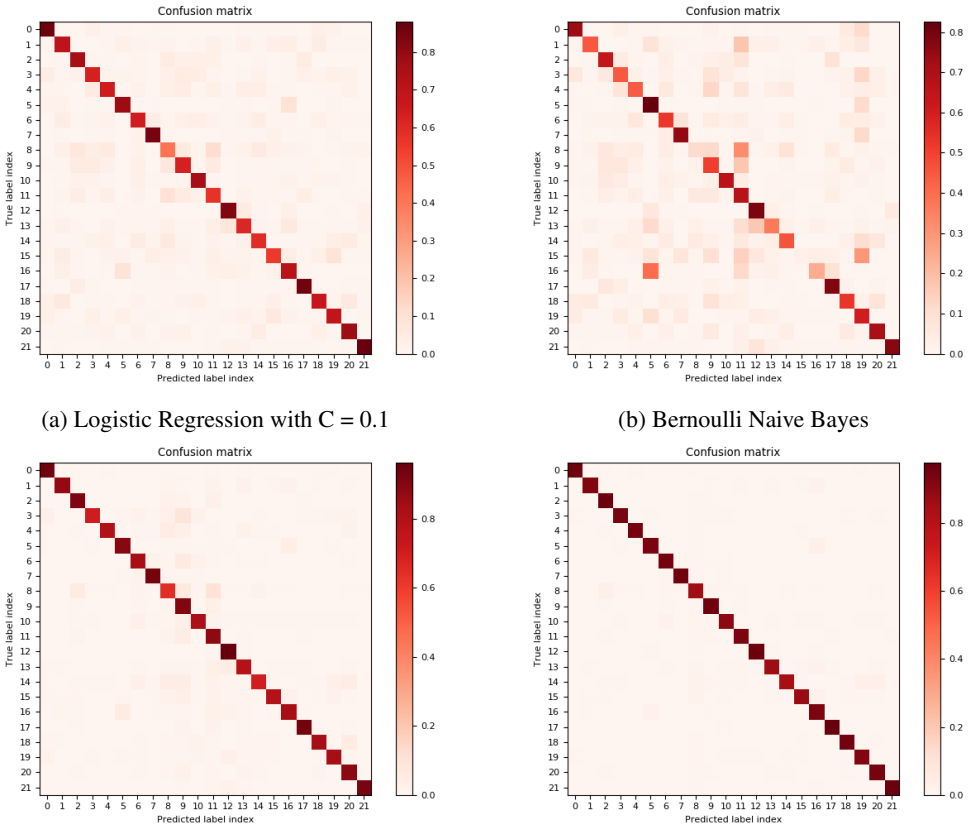
6.5 Performance on the Test Set

The best model was chosen to be the second CNN model described in Section 5.4, CNN_2, trained with a batch size of 100 over 10 epochs. This model was applied to the test set, which was unseen up till this point, and the results obtained were as follows: Train Accuracy = 0.9370, Train Log Loss = 2.1749, Development Accuracy = 0.9301, and Development Log Loss = 2.4142.

Figure 4 shows the confusion matrix.

b_{CNN_2}	e_{CNN_2}	Train Log Loss	Log Loss	Accuracy	Precision	Recall
10	1	4.4667	4.4818	0.8702	0.88	0.87
10	3	7.0013	7.0585	0.7956	0.83	0.80
10	10	17.0969	17.0190	0.5072	0.56	0.51
10	50	21.8218	21.7504	0.3703	0.42	0.37
50	1	3.1316	3.2136	0.9069	0.91	0.91
50	3	2.7945	2.9327	0.9151	0.92	0.91
50	10	3.1560	3.2439	0.9061	0.91	0.91
50	50	7.8106	7.7712	0.7750	0.80	0.77
100	1	3.3828	3.4111	0.9012	0.90	0.90
100	3	2.5201	2.6803	0.9224	0.92	0.92
100	10	2.1749	2.3804	0.9311	0.93	0.93
100	50	2.7852	2.9440	0.9148	0.92	0.91

Table 6: Results using different batch size and epochs for CNN_2 architecture



(a) Logistic Regression with C = 0.1

(b) Bernoulli Naive Bayes

(c) MLP with batch size = 100 and epochs = 3

(d) CNN with batch size = 100 and epochs = 10

Figure 3: Confusion matrices

7 Discussion

The performance of the Naive Bayes models was the poorest. Being predominantly a text classifier, it was unable to understand the subtleties of image data. As seen from the confusion matrix, Figure 3b, there were certain classes that were misclassified by the model very frequently. It was interesting to observe the types of mistakes made by this model, for example, it predicted a large number of pencil images as candles and a large number of peanut images as snowman. These pairs have similar traits

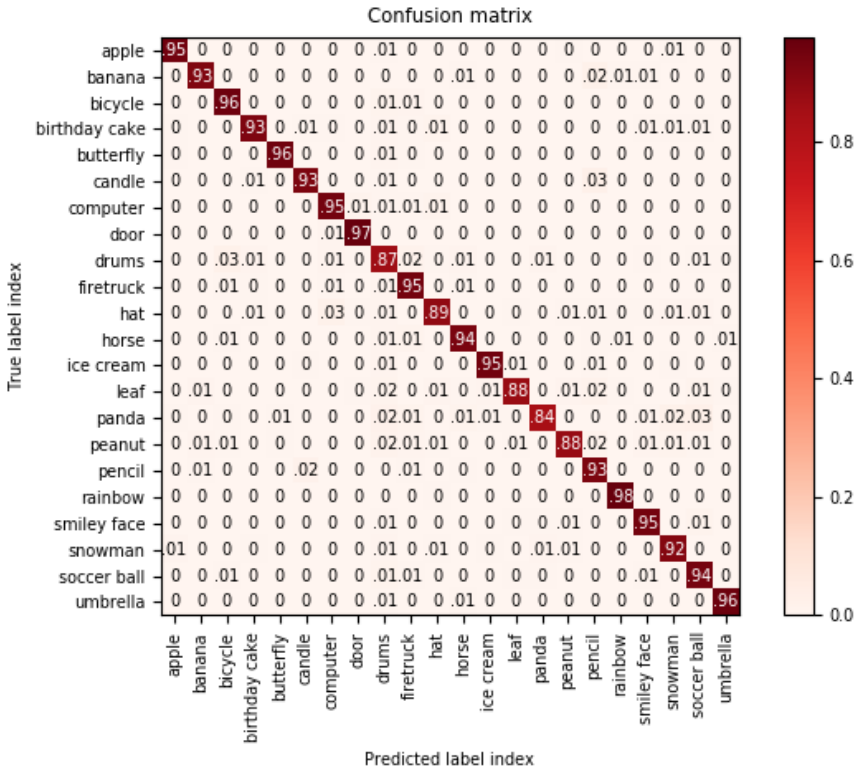


Figure 4: Confusion Matrix obtained on test set

when drawn that could be confusing even to the human eye, so it is very interesting to see the model face similar challenges.

Logistic regression performed surprisingly well given the simplicity of the model. It is clear from the results in Table 2 and the confusion matrix, Figure 3a, that this model was able to accurately predict most of the classes. Additionally, there was no single class on which the model performed significantly worse than the others, unlike what was seen with Naive Bayes.

The MLP model performed better than the traditional models and achieved good accuracy. It is known that MLP models have a tendency to over-fit and therefore we tried to use dropout layers to curb this behavior. Despite that, some over-fitting was observed in all the experiment runs as the model consistently performed better on the training set than the development set. Additionally, when the MLP model flattens the image array it loses information regarding the spacial arrangement of pixels. CNN models are able to overcome this by keeping the image as a 2-dimensional matrix for the convolutional layers.

The CNN models performed the best on our dataset obtaining accuracy numbers surprisingly close to our oracle! The CNN method looks at an image, piece by piece, simplifying the complex information contained in images. This allows it to understand the nuances of an image like edges and corners which makes the method a widely accepted choice for image classification and our results align with this notion.

Testing the model on a set that was not seen during the training and development process allowed us to verify that we were not over-fitting on either our training set or our development set. The model achieved similar log loss and accuracy on all three parts of the dataset.

8 Demonstration

To demonstrate this project, we created an application using the OpenCV library on Python [6] which creates a canvas, allowing the user to draw on it. This drawing is then compressed into a 28×28

gray-scale bitmap to match the training data. The best model obtained from our experimentation is then applied to the compressed image to make a prediction about its class.

We observed that the demonstration application predicts the classes with a significantly lower accuracy of $\sim 75\%$, compared to the 93% accuracy observed on the test set. We attribute this to the difference in the method of image compression between the training data and the input created by the demo app.

An interesting observation made from the demo app was that it was never able to recognize a smiley face. If a smiley face was drawn with a circle around it, it was classified as soccer ball and if a face was drawn without a circle, it was classified as panda. This gives us some insight into the kind of features that were learned by the CNN, like a circle and a face, but it was not able to combine the two to recognize a smiley face.

9 Future Work

Although we were able to achieve accuracy scores close to our oracle with the chosen model, there is a lot of scope for improvement. The first thing we would like to do is tune more hyper-parameters in the CNN model. This includes the number of filters in the convolutional layers, the number of neurons in the fully-connected layers, the size of the max-pooling filters and stride, as well as dropout probabilities.

For the demonstration application, we can work on improving the image compression technique to better match the training data. This should allow the application to identify drawings with higher accuracy.

Eventually, we should be able to expand this model to take into consideration all the classes available on the Google Quickdraw dataset [3], allowing it to identify any drawing and not be limited to 22 classes.

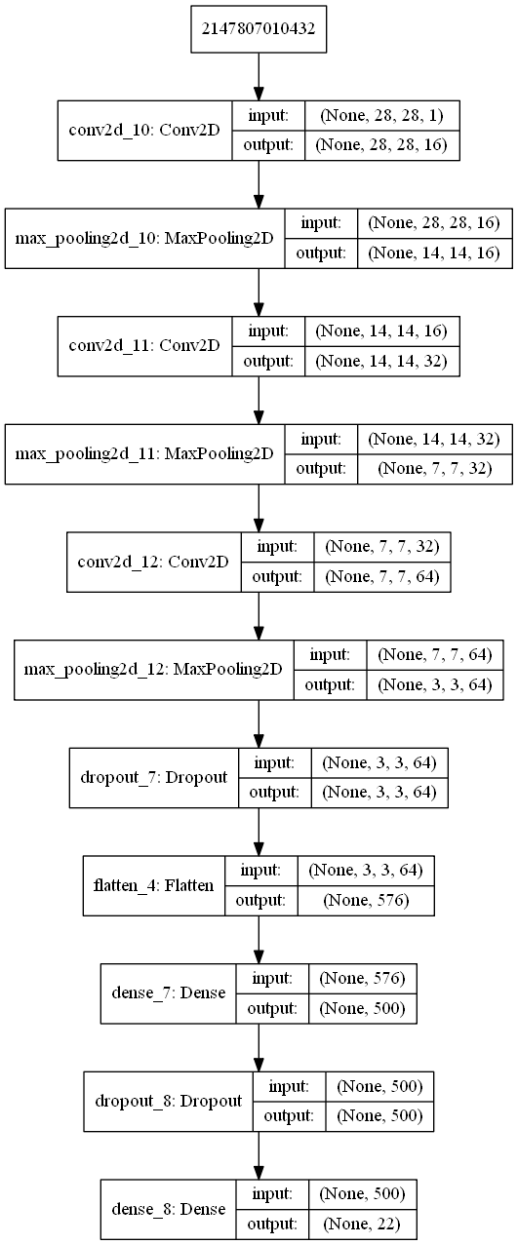
10 Conclusion

Through our experimentation using the dataset of sketch drawings, we observed various challenges that arise with drawing classification. Looking at photographs, objects are more easily identifiable than with drawing data, where the pictures tend to be more abstract and the data varies tremendously based on the artists' interpretation of as object as well as their art style. In addition to this, data compression can lead to the loss of valuable characteristics of a drawn image.

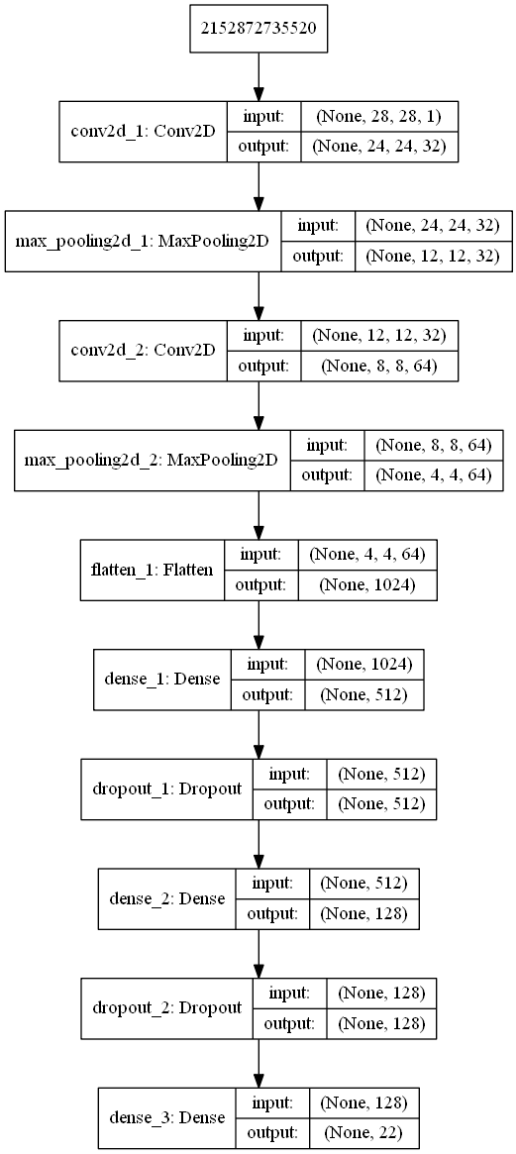
From our survey of different classification methods we see that CNN performs the best on the dataset. We also observe that deep learning methods perform better than traditional methods in general, achieving higher accuracy in identifying the free-hand drawings. Our final results on the test set show that the chosen CNN model is able to classify the drawing with accuracy scores close to the human classifier results obtained as our oracle.

Our source code for this project can be found on `https://github.com/adutta158/CS221ProjectSp19`.

Appendix



(a) CNN architecture 1



(b) CNN architecture 2

Figure 5: CNN Architectures

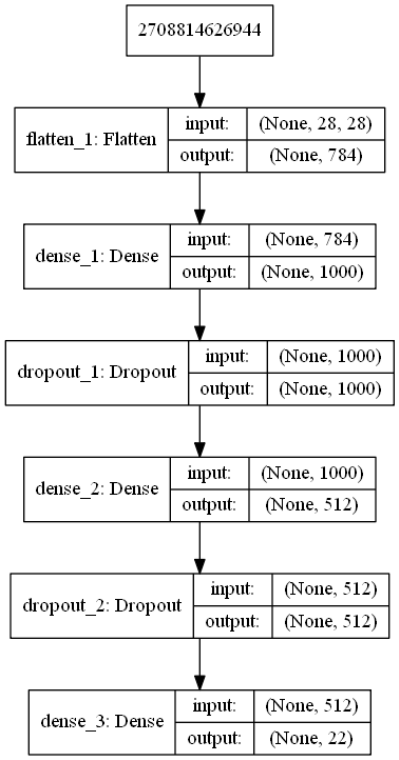


Figure 6: MLP architecture

References

[1] Guo, T., Dong, J., Li, H. & Gao, Y. (2017) Simple convolutional neural network on image classification. *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pp. 721–724.

[2] Ha, D. & Eck, D. (2017) A neural representation of sketch drawings. arXiv:1704.03477.

[3] Jongejan, J., Rowley, H., Kawashima, T., Kim, J. & Fox-Gieg, N. (2016) The Quick, Draw! - A.I. Experiment. `https://quickdraw.withgoogle.com/data`.

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

[5] Chollet, F. (2015) Keras. `https://keras.io`.

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