CS-UY 4563: Machine Learning:

Final Project Written Report

Multi-Class Classification of Animal Sketches

11:00AM Section

April 26, 2022

Professor Linda N. Sellie

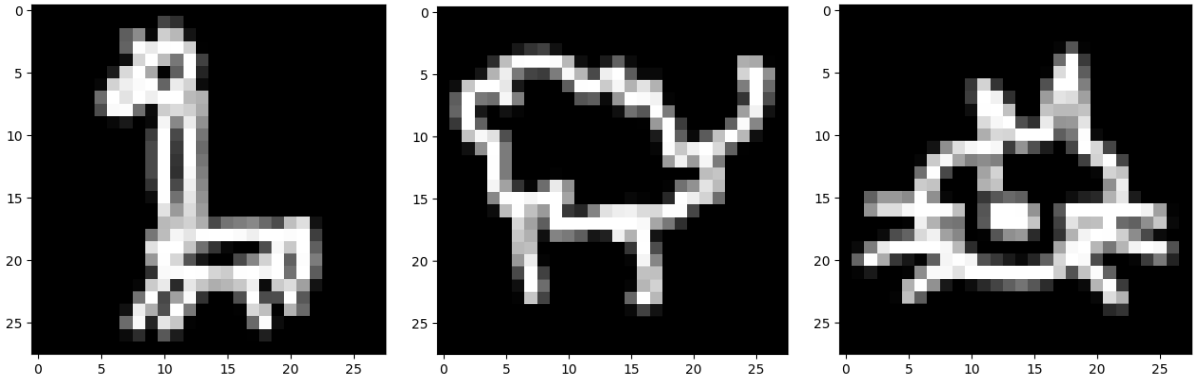
Berry Liu, Thaison Le

1. Introduction

This project focuses on using different machine learning algorithms to identify and classify drawings of animals. The dataset was retrieved from Google's Quick Draw dataset and is composed of bitmaps of drawings made by users for that category. Approximately 100,000 samples were available for each classification, however we limited the sample size to 2000 per category for training purposes.

In our project, we utilized logistic regression, support vector machines, and neural networks to develop classifiers for drawings of the categories of sheep, giraffe and cats. All three models were optimized through multiple iterations of the algorithm and tweaking hyperparameters, feature transformations and normalization techniques.

Randomly selected examples from each class (from left to right: giraffe, sheep, cat)



2. Data Preprocessing

We chose three classes out of the QuickDraw dataset: sheep, giraffe, and cat. We selected 1000-2000 examples from each class out of the approximately 100,000 examples available for each class. The dataset contained many outliers, likely due to the variability in people's drawing skills, which presented challenges for our model's accuracy.

To prepare the dataset for our experiments, we divided the 2000 samples per class into an 80-10-10 split for training, validation, and testing purposes, respectively. Each example in the dataset was represented as a 28x28 numpy bitmap, with the drawing centered on the bitmap. The bitmap was already flattened to a one-dimensional array containing 784 features. The array values ranged from 0 to 255, with 0 representing a blank pixel and 255 indicating a fully drawn pixel.

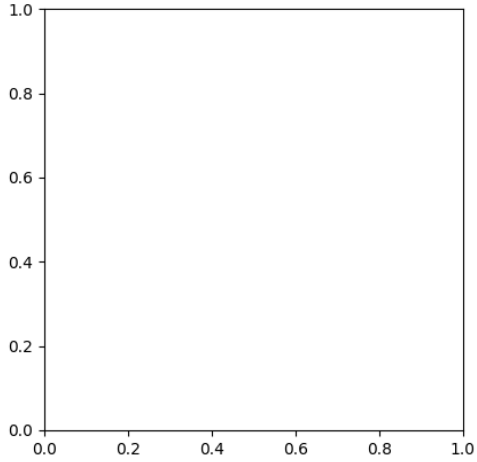
To normalize each feature, we ran the algorithm with default values of hyperparameters on the control dataset, as well as copies of the dataset normalized using the min/max normalization and standardization techniques. In the tests, standardization resulted in the highest accuracy, so we normalized the features using standardization before running the algorithms.

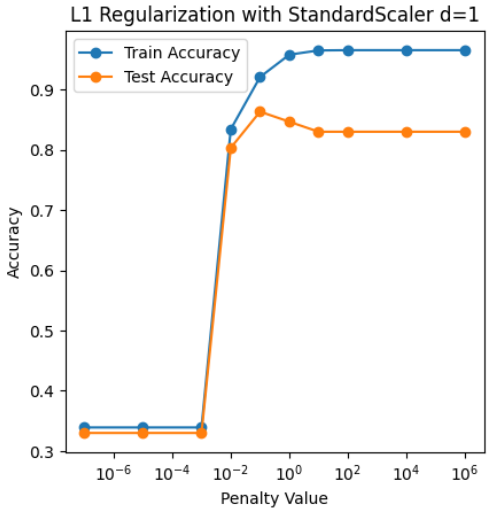
3. Models

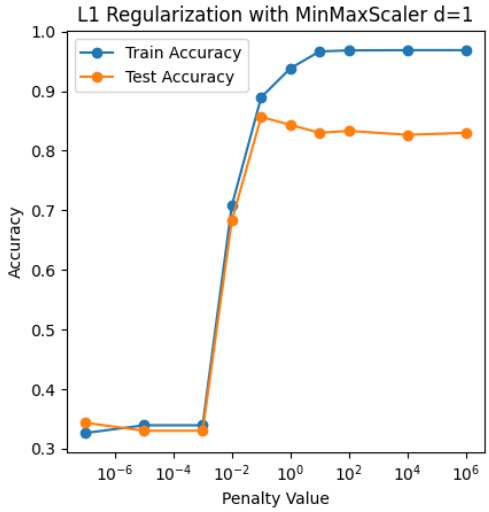
Logistic Regression:

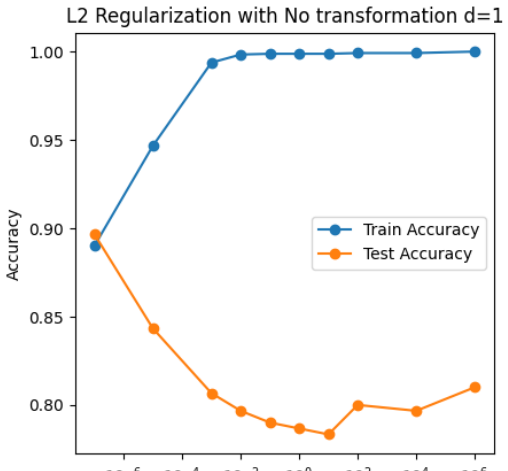
The first model that we prepared was Logistic Regression. We chose a dataset of 1000 for this model, as training took a lot of resources. We used SkLearn's implementation of Logistic Regression. For choosing the optimal model, we chose to permutations of the following hyperparameters and feature transformations: scaling the data(No scaling, StandardScaler, MinMaxScaler), raising the data to degree 1 and 2, and testing L1 and L2 regularization using hyperparameters of C = [0.0000001, 0.00001, 0.001, 0.01, 0.1, 1, 10, 100, 10000, and 1000000]. This gave us a total of 3 * 2 * 2 = 12-2 = 10 (minus 2 as the solver, stochastic gradient descent, took immensely long to train on unscaled data) unique models. For feature transformations, we used SkLearn's preprocessing library.

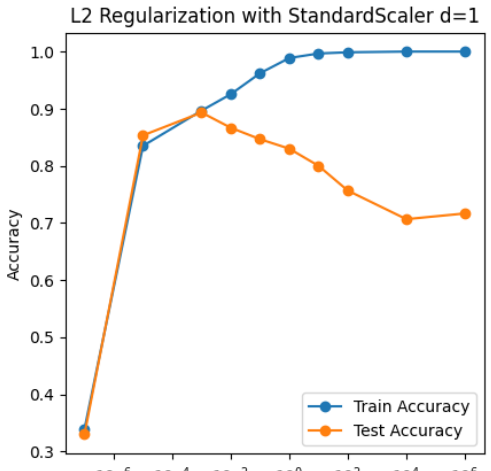
For each model, we validated our results with a validation set. We then plotted the training and validation accuracies below:

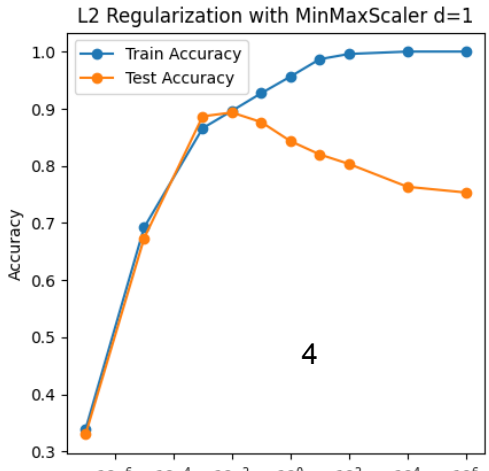


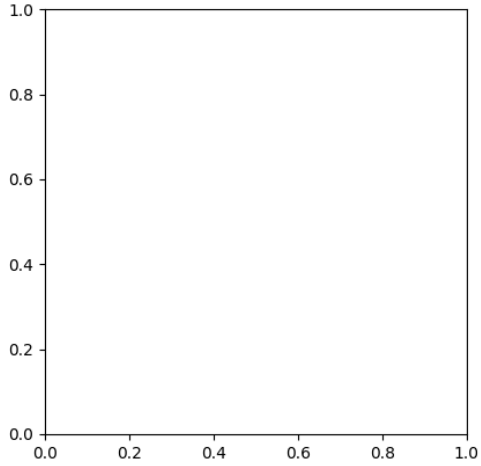


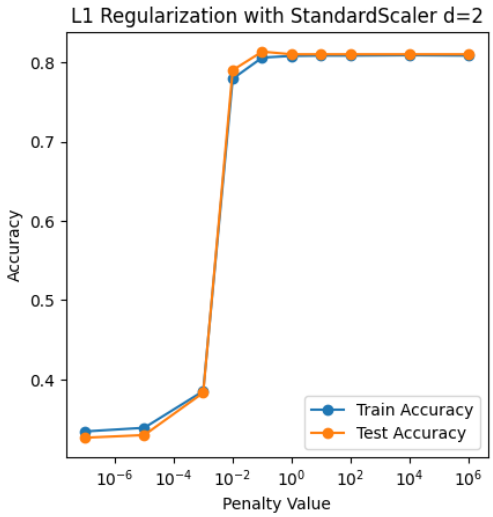


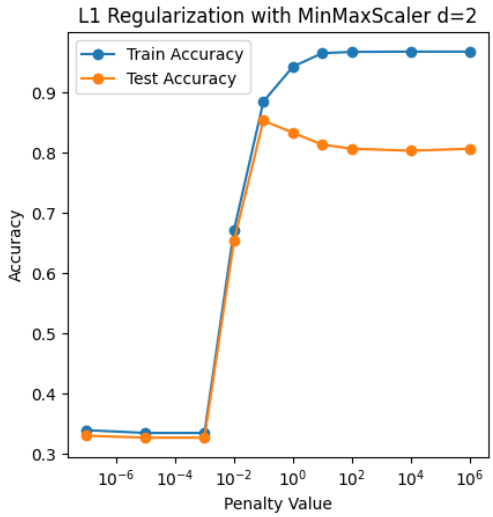


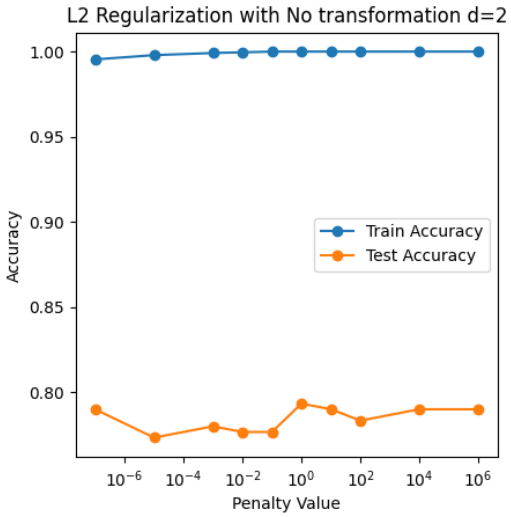


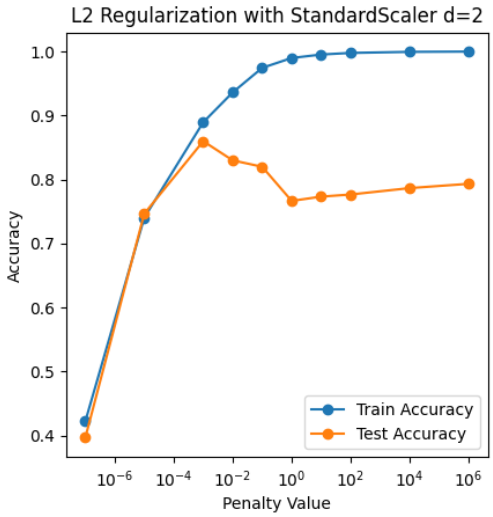


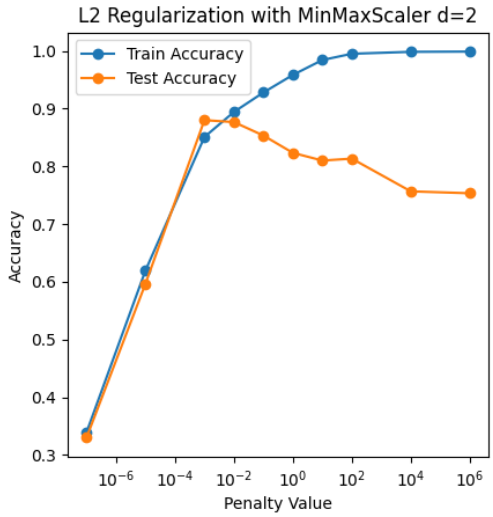



L1 Regularization with StandardScaler d=2
Accuracy
Penalty Value
Train Accuracy
Test Accuracy

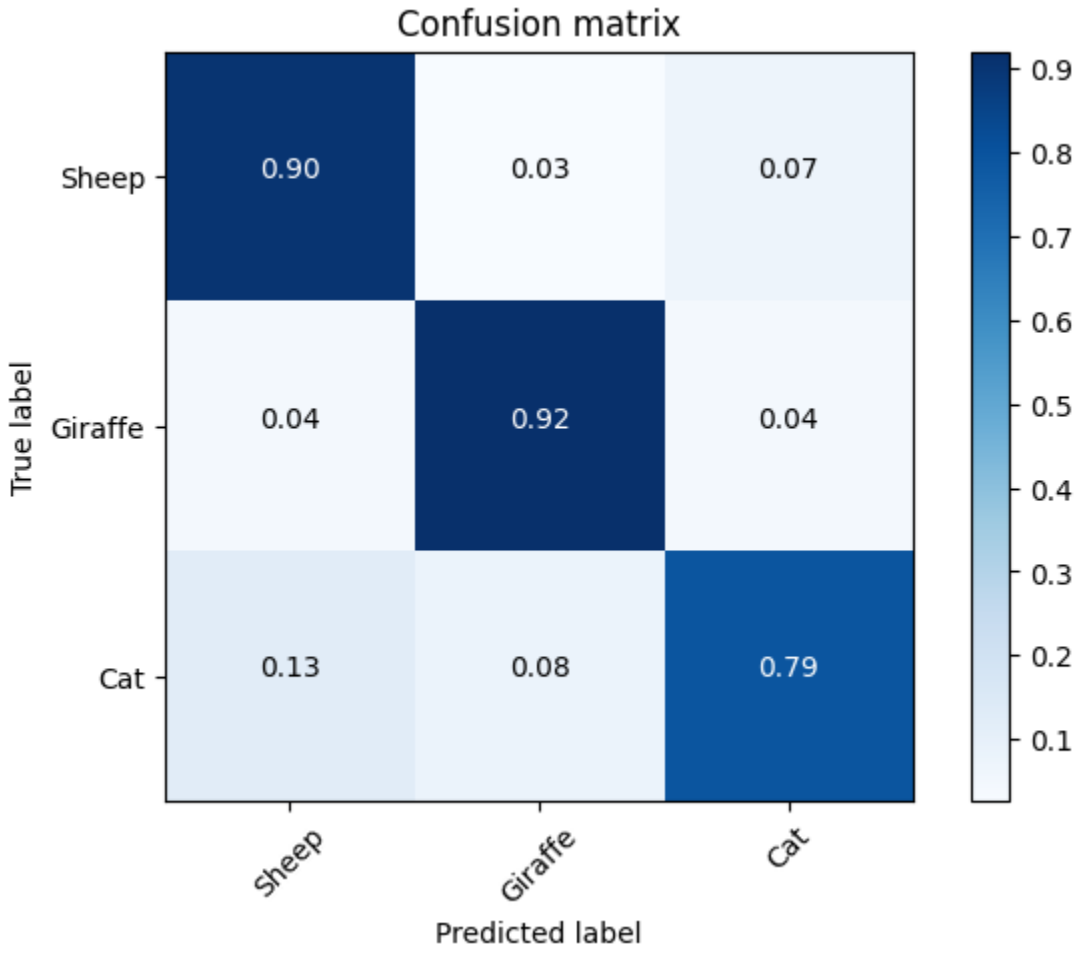

L1 Regularization with MinMaxScaler d=2
Accuracy
Penalty Value
Train Accuracy
Test Accuracy


L2 Regularization with No transformation d=2
Accuracy
Penalty Value
Train Accuracy
Test Accuracy


L2 Regularization with StandardScaler d=2
Accuracy
Penalty Value
Train Accuracy
Test Accuracy


L2 Regularization with MinMaxScaler d=2
Accuracy
Penalty Value
Train Accuracy
Test Accuracy

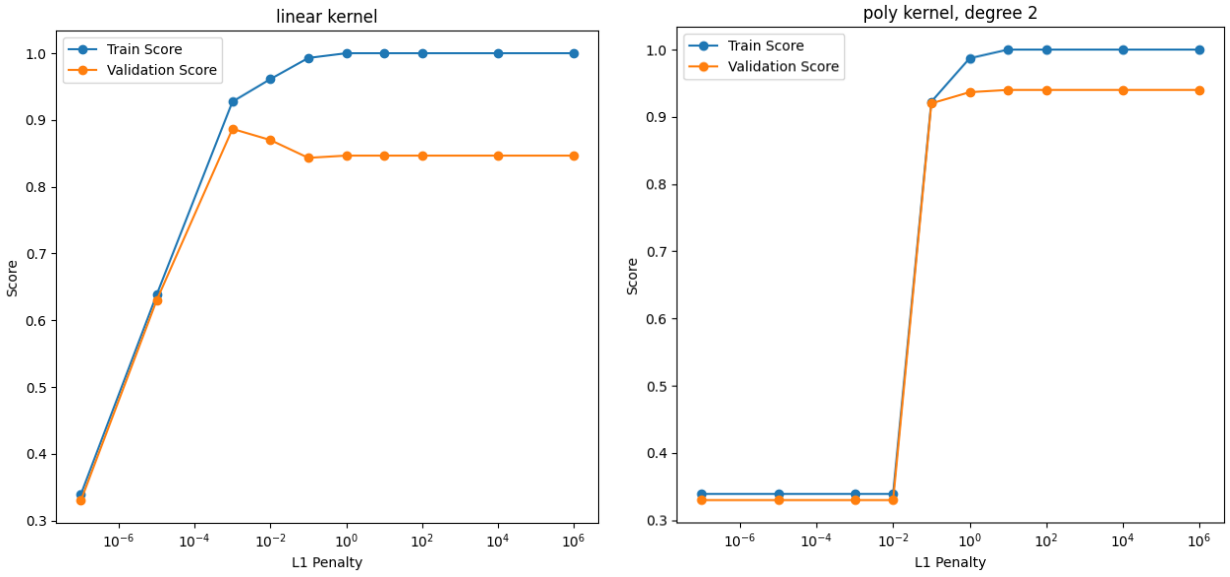
We then evaluated the best model on the test set. The confusion matrix for the best model - L2 Regularization with MinMaxScaling and degree 1 with C_val = 0.01 - is plotted below:

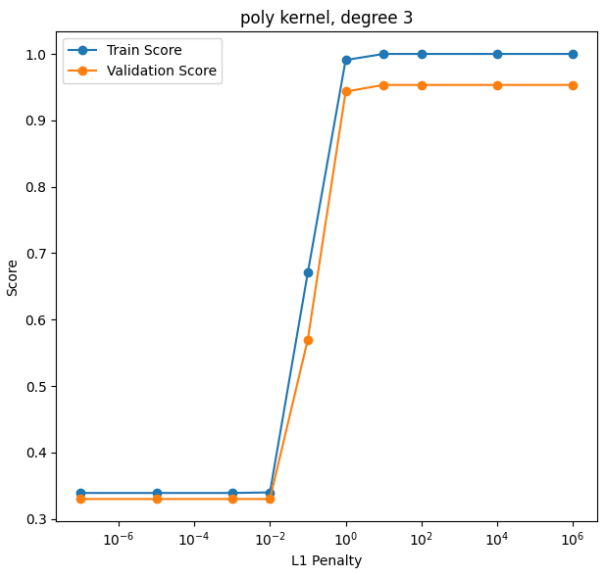


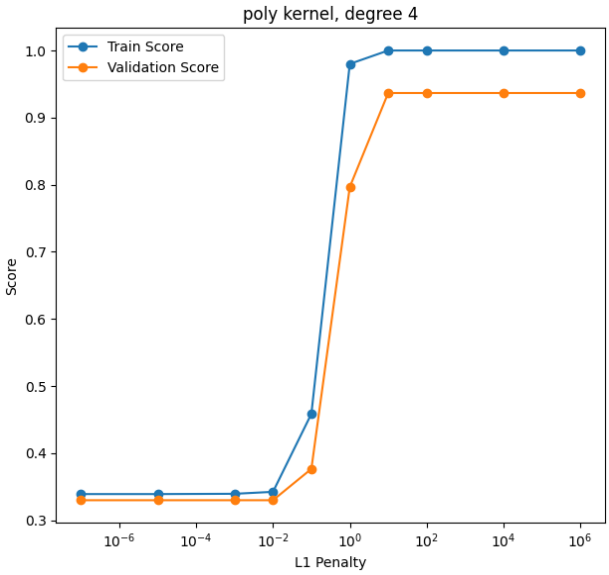
An observed trend in the confusion matrix is that the cat model is predicted as a sheep often. This is likely due to the fact that they have a similar shape.

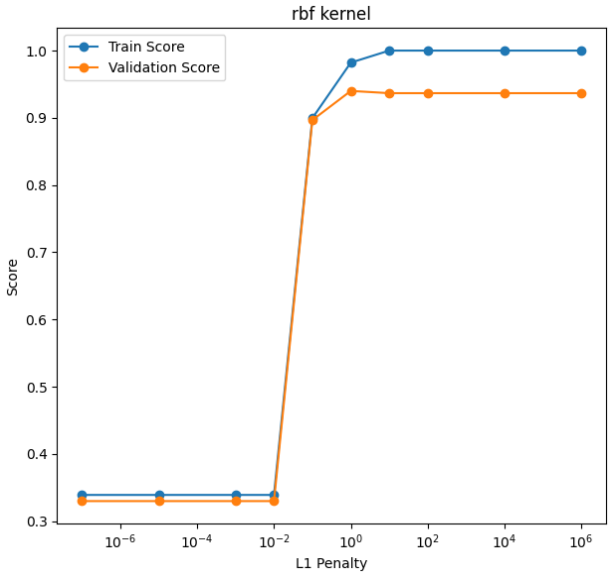
Support Vector Machines

The next machine learning algorithm, we modeled our dataset using was using Support Vector Machines. To run the algorithm, we used the svm functions provided by the sklearn library. The processed set of data (including normalization via standardization) was used to train all of the SVM models. The hyperparameters we chose to experiment with included the type of kernel (linear, polynomial of degrees 2,3,4 and radial basis, as well as the hyperparameter for Ridge Regression squared using C values = [0.0000001, 0.00001, 0.001, 0.01, 0.1, 1, 10, 100, 10000, and 1000000]. We then plotted the test and train accuracies of each feature transformation/ kernel functions for each C value on the same graph, using a logarithmic x-axis to separate the X values with equal spacing. The resulting graphs are shown below:

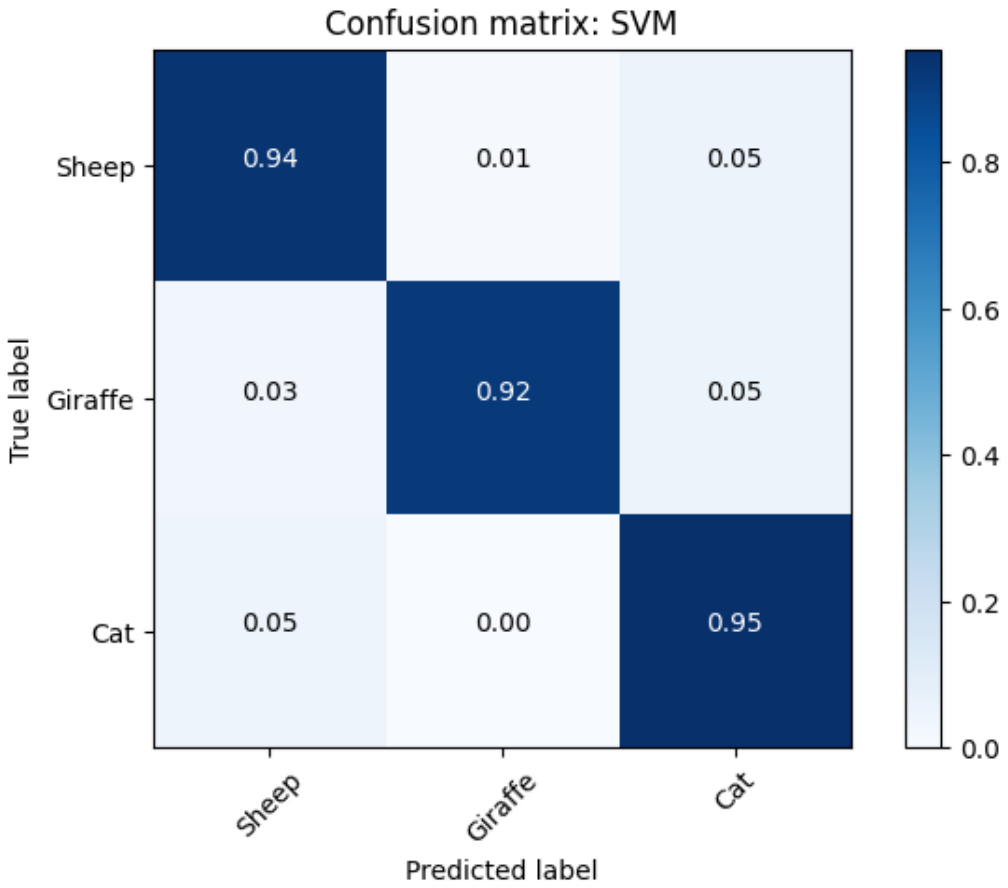






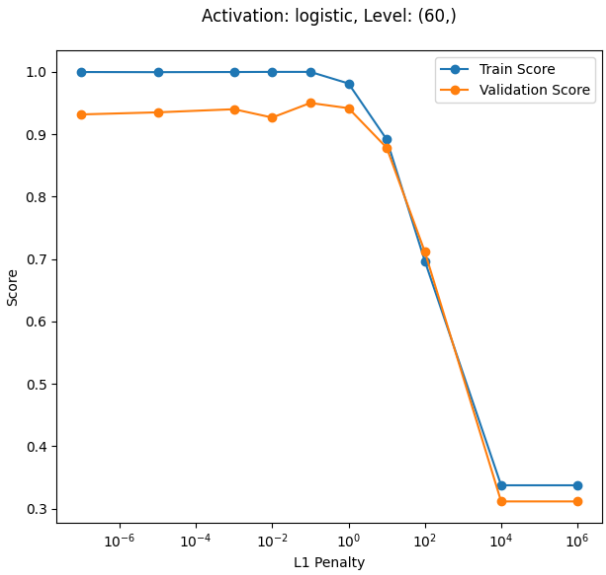


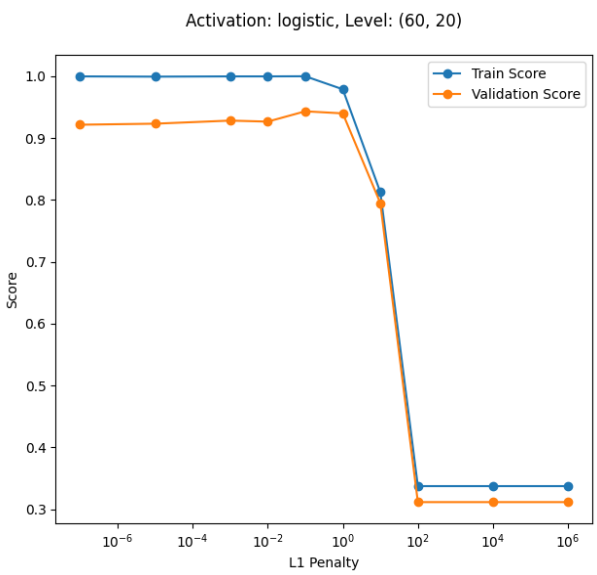
We selected the model that resulted in the highest test accuracy of 0.953, which ended up having the hyperparameters: kernel=polynomial, C=10, degree = 3, scaler=StandardScaler. We used that model to create the confusion matrix, which is shown below.

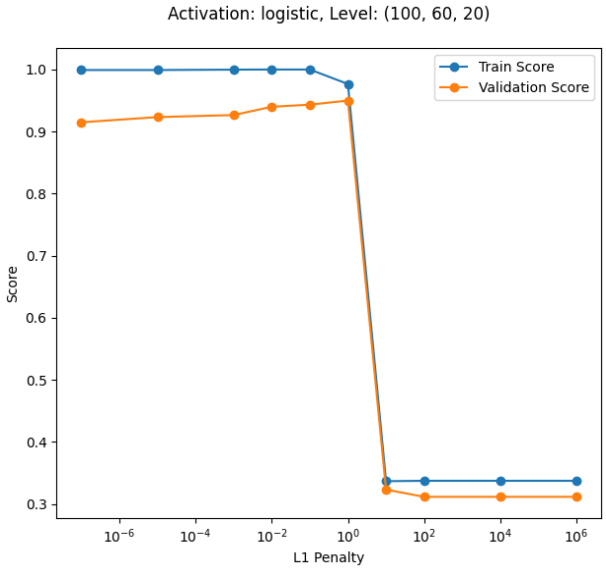


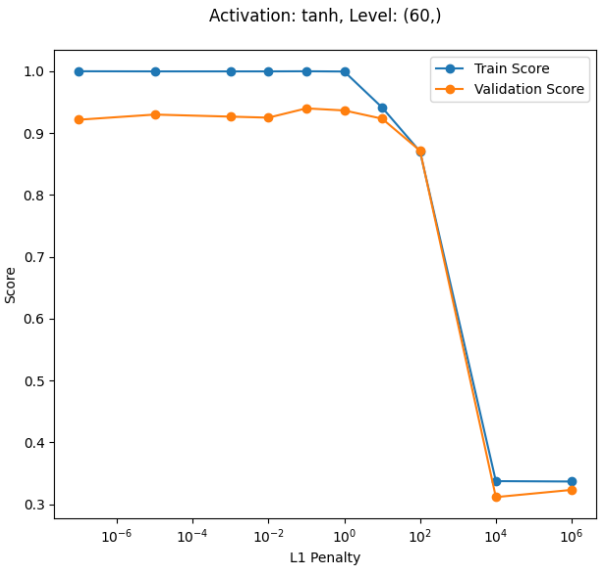
Neural Networks

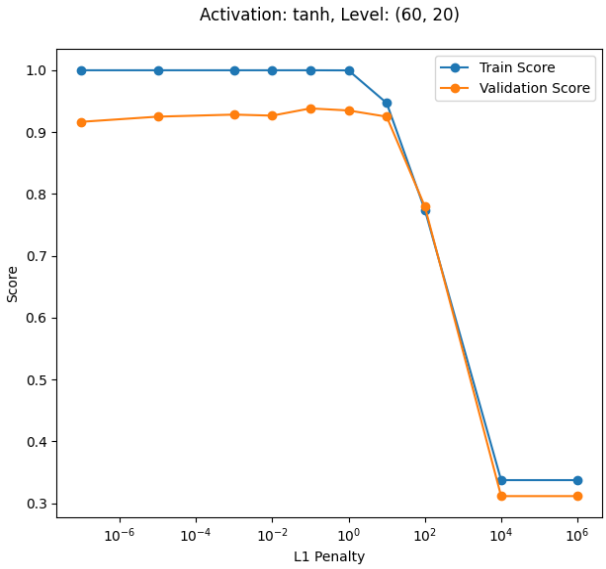
As our last model, we trained our dataset on neural networks. In training the model, we used the A number of hyperparameters that were tested to optimize the dataset included activation functions, the network architecture (hidden layer networks), as well as hyperparameters for Ridge Regression squares, or the C values. The activation functions that were trained were Rectified Linear Unit (ReLu), logistic, and tanh. Three different neural network architectures were tested: two hidden layers of neurons (60 and 20) , three hidden layers of neurons(100, 60 and 20), and one hidden layer of neurons (60). Each configuration was tested using the C-Values [0.0000001, 0.00001, 0.001, 0.01, 0.1, 1, 10, 100, 10000, and 1000000]. With those hyperparameters in mind, we plotted the testing and training accuracies for different values of each hyperparameters for each C value on each graph in the same format as the previous. The resulting graphs are shown below:

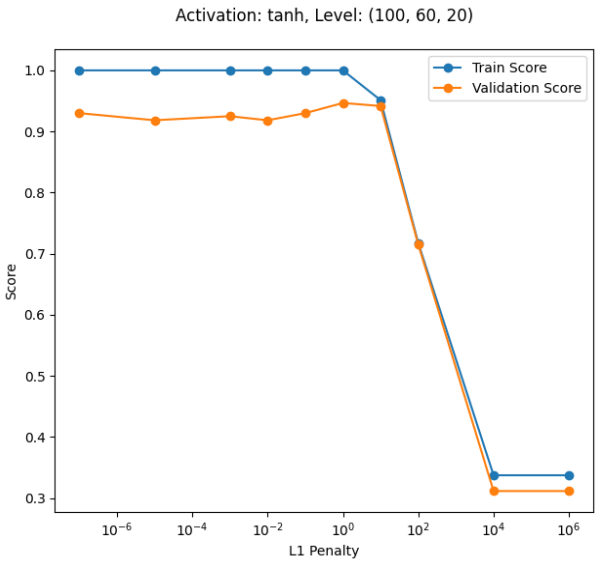

Activation: logistic, Level: (60,)
Train Score
Validation Score
Score
L1 Penalty

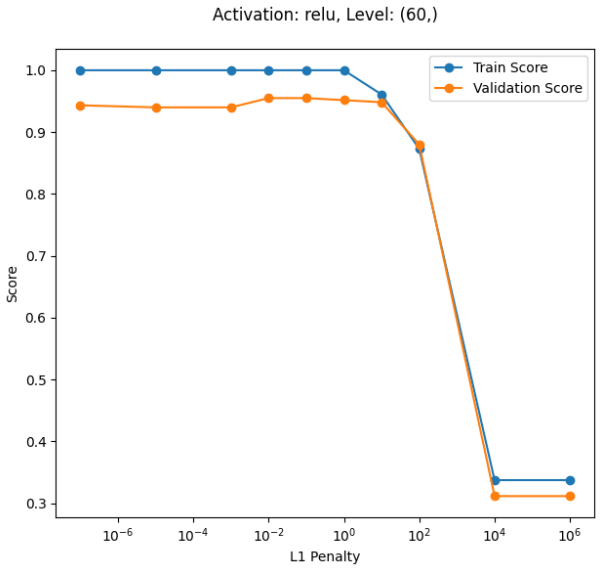

Activation: logistic, Level: (60, 20)
Train Score
Validation Score
Score
L1 Penalty

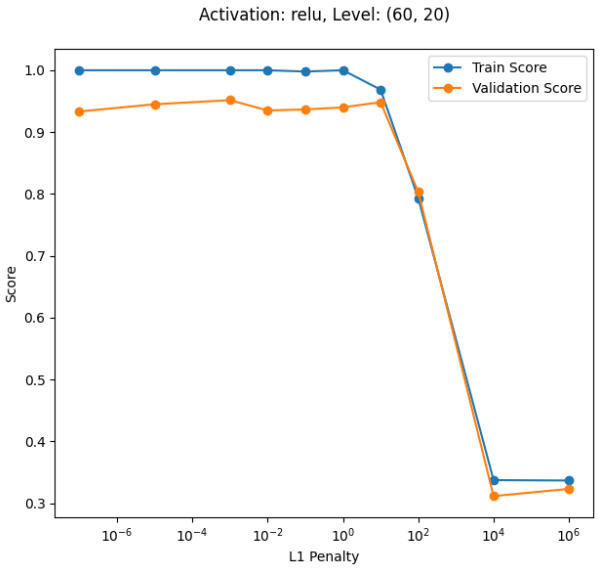

Activation: logistic, Level: (100, 60, 20)
Train Score
Validation Score
Score
L1 Penalty

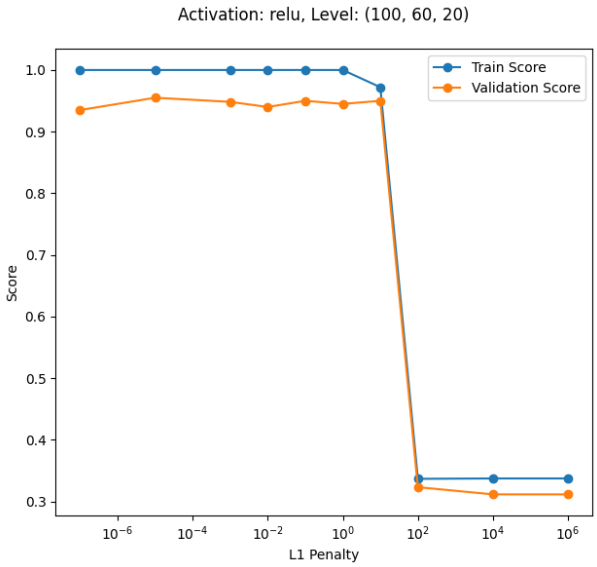

Activation: tanh, Level: (60,)
Train Score
Validation Score
Score
L1 Penalty


Activation: tanh, Level: (60, 20)
Train Score
Validation Score
Score
L1 Penalty

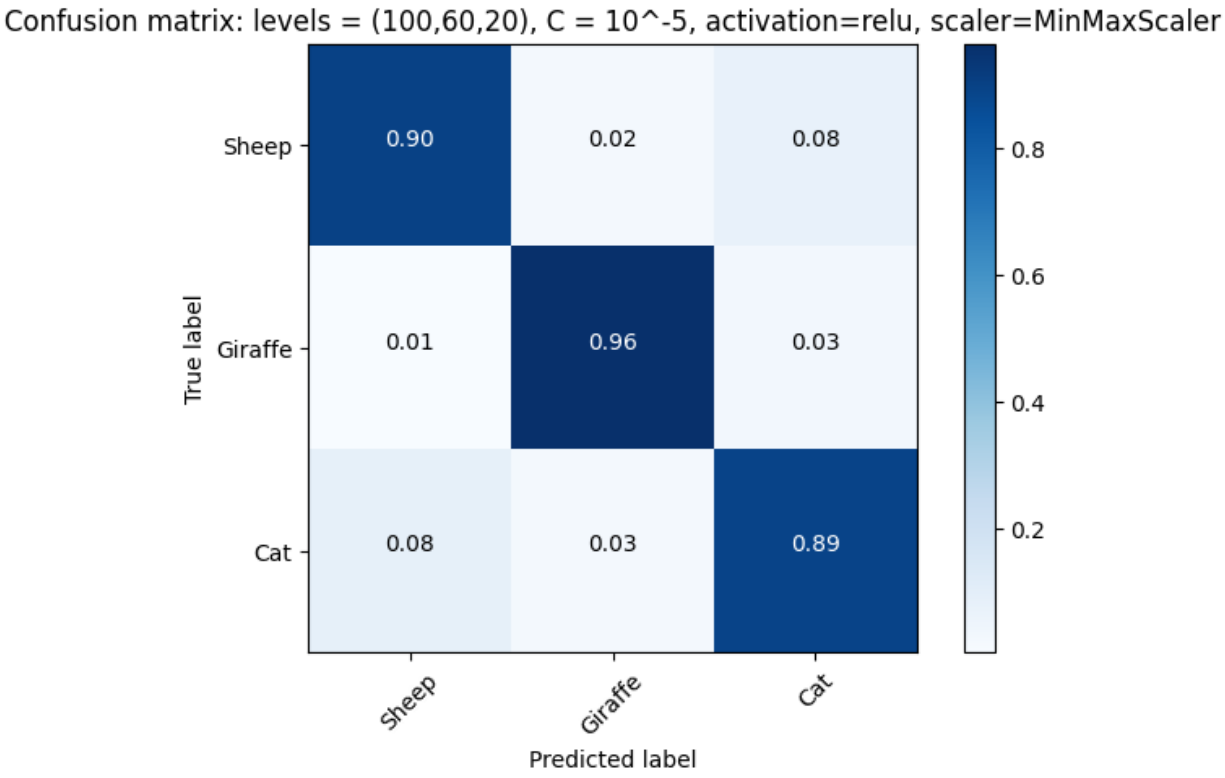

Activation: tanh, Level: (100, 60, 20)
Train Score
Validation Score
Score
L1 Penalty


Activation: relu, Level: (60,)
Train Score
Validation Score
Score
L1 Penalty


Activation: relu, Level: (60, 20)
Train Score
Validation Score
Score
L1 Penalty



We selected the model that resulted in the highest test accuracy, which were the hyperparameters: levels = (100,60,20), C = 10^-5, activation=relu, scaler=MinMaxScaler. We used those hyperparameters to retrain on the model and create a Confusion matrix for the result of our best hyperparameters.



4.Tables

Logistic No Regularization

	No Transformation	Standard Scaler	MinMax Scaler
Train	1	1	0.99875
Validation	.79	.78	.743333

Logistic L1 and L2 Regularization

C -> Feature Transform↓	1e-7	1e-5	1e-3	1e-2	1e-1	1	1e1	1e2	1e4	1e6
No Scaler L2 d(degree)=1 Train	0.89	0.9467	0.9938	0.9983	0.9988	0.9988	0.9988	0.9992	0.9992	1.0
No Scaler L2 d(degree)=1 Val	.8967	.8433	.8067	.7967	.79	.7867	.7833	.8	.7967	.81
Standard Scaler L1 d=1 Train	0.3392	0.3392	0.3392	0.8342	0.9208	0.9579	0.965	0.9654	0.9654	0.9654
Standard Scaler	.33	.33	.33	.8033	.8633	.8467	.83	.83	.83	.83

L1 d=1 Val										
Standard Scaler L2 d=1 Train	0.3392	0.835	0.8967	0.9254	0.9621	0.9888	0.9967	0.9988	1.0	1.0
Standard Scaler L2 d=1 Val	.33	.8533	.8933	.8667	.8467	.83	.8	.7567	.7067	.7167
MinMax Scaler L1 d=1 Train	0.3263	0.3392	0.3392	.7083	.8896	.9383	.9667	.9683	.9688	.9688
MinMax Scaler L1 d=1 Val	.3433	.33	.33	.6833	.8567	.8433	.83	.8333	.8267	.83
MinMax Scaler L2 d=1 Train	0.3392	.6921	.8658	.8963	0.9271	.9563	0.9867	0.9558	1.0	1.0
MinMax Scaler L2 d=1 Val	.33	.6733	.8867	.8933	.8767	.8433	.82	.8033	.7633	.7533
No Scaler L2 d(degree)=2	1.0	.9954	.9979	.9992	.9996	1.0	1.0	1.0	1.0	1.0

train										
No Scaler L2 d(degree)=2 test	.79	.79	.7733	.78	.7767	.7767	.7933	.79	.7833	.79
Standard Scaler L1 d=2 Train	0.3392	.3346	.3854	.8892	.8058	.8079	.8088	.8088	.8083	.8088
Standard Scaler L1 d=2 Val	.33	.3267	.3833	.7933	.81	.81	.81	.81	.8	.81
Standard Scaler L2 d=2 Train	1.0	.4421	.7404	.8896	.9363	.9746	.99	.9954	.9979	.9996
Standard Scaler L2 d=2 Val	.78	.3967	.7467	.86	.83	.82	.7667	.7733	.7767	.7867
MinMax Scaler L1 d=2 Train	.3392	.3392	.3392	.6708	.885	.9429	.9658	.9671	.9679	.9679
MinMax Scaler L1 d=2 Val	.33	.33	.33	.6533	.8533	.8333	.8133	.8067	.8033	.81
MinMax Scaler	.9988	.3392	.6196	.8508	.8942	.9279	.9583	.9842	.995	.9983

L2 d=2 Train										
MinMax Scaler L2 d=2 Val	.7433	.33	.5967	.88	.8767	.8533	.8233	.81	.8133	.7567

Support Vector Machines L2^2 Regularization

C -> Feature Transform↓	1e-7	1e-5	1e-3	1e-2	1e-1	1	1e1	1e2	1e4	1e6
Linear Train	0.638	0.638	0.928	0.961	0.993	1	1	1	1	1
Linear Test	0.33	0.63	0.887	0.87	0.843	0.847	0.847	0.847	0.847	0.847
Polynomial Degree 2 Test	0.339	0.339	0.339	0.339	0.923	0.987	1	1	1	1
Polynomial Degree 2 Train	0.33	0.33	0.33	0.33	0.92	0.937	0.94	0.94	0.94	0.94
Polynomial Degree 3	0.339	0.339	0.339	0.34	0.671	0.991	1	1	1	1

Test										
Polynomial Degree 3 Train	0.33	0.33	0.33	0.33	0.57	0.943	0.953	0.953	0.953	0.953
Polynomial Degree 4 Test	0.339	0.339	0.34	0.343	0.459	0.98	1	1	1	1
Polynomial Degree 4 Train	0.33	0.33	0.33	0.33	0.377	0.797	0.937	0.937	0.937	0.937
Radial Basis Function Train	0.339	0.339	0.339	0.339	0.9	0.982	1	1	1	1
Radial Basis Function Test	0.33	0.33	0.33	0.33	0.897	0.94	0.937	0.937	0.937	0.937

Neural Networks L2 Regularization

C -> Layers ↓	1e-7	1e-5	1e-3	1e-2	1e-1	1	1e1	1e2	1e4	1e6
Logistic (60,) Train	0.999	0.999	0.999	0.999	0.997	0.981	0.894	0.717	0.337	0.337
Logistic (60,)	0.938	0.918	0.937	0.931	0.948	0.95	0.89	0.727	0.323	0.323

Test										
Logistic (60,20) Test	0.999	0.999	0.999	0.999	1	0.982	0.790	0.337	0.337	0.337
Logistic (60,20) Train	0.923	0.928	0.921	0.94	0.943	0.95	0.778	0.312	0.312	0.312
Logistic (100, 60,20) Train	0.999	0.998	0.999	1	1	0.975	0.337	0.337	0.337	0.337
Logistic (100, 60,20) Test	0.913	0.917	0.925	0.947	0.941	0.935	0.323	0.323	0.311	0.311
Tanh (60,) Train	0.999	0.999	0.999	0.999	1	0.999	0.941	0.881	0.337	0.337
Tanh (60,) Test	0.93	0.92	0.93	0.932	0.952	0.95	0.933	0.875	0.323	0.312
Tanh (60,20) Test	0.999	0.999	1	1	1	1	0.947	0.773	0.337	0.337
Tanh (60,20) Train	0.922	0.922	0.92	0.927	0.938	0.943	0.938	0.77	0.312	0.312

Tanh (100, 60,20) Train	1	1	1	1	0.999	1	0.953	0.729	0.337	0.337
Tanh (100, 60,20) Test	0.908	0.933	0.933	0.932	0.935	0.945	0.935	0.728	0.323	0.311
ReLu (60,) Train	1	1	1	1	1	0.999	0.959	0.878	0.337	0.337
ReLu (60,) Test	0.943	0.94	0.942	0.945	0.953	0.953	0.95	0.883	0.312	0.323
ReLu (60,20) Test	1	1	1	1	0.991	1	0.967	0.810	0.337	0.337
ReLu (60,20) Train	0.942	0.942	0.943	0.955	0.943	0.952	0.947	0.812	0.312	0.323
ReLu (100, 60,20) Train	1	1	1	1	1	1	0.971	0.337	0.337	0.337
ReLu (100, 60,20) Test	0.935	0.94	0.945	0.942	0.938	0.952	0.958	0.312	0.312	0.312

5. Conclusion

The test accuracies for each model were fairly high, in the range of 90% accuracy. This is most likely due to quality and amount of data.

For logistic regression, the highest observed test accuracy was 89.6%. This was achieved using MinMax Scaling feature transformation on the data and L2 regularization with a C value of 0.01. Some interesting trends were that feature transformation increased the accuracy of the model across the board, with similar increases from both MinMax and standard scaling. MinMax recorded the highest accuracy, but it could easily have been Standard scaling if tested again. Squaring the data did not help the model. In fact, it actually decreased each model's accuracy on new data. No regularization models performed much worse than models with regularization. Among the C values, the sweet spot for almost all models was around 10^-2. This provided the highest accuracy in the validation set. This was most likely due to the fact that although most people's drawings were generally similar, there is still a decent amount of variance between drawing skills, so the model had to reduce it by using a decent bit of regularization. In some models, training accuracy hit 1.0, meaning 100% training accuracy, yet validation accuracy decreased, indicating overfitting was occurring.

For SVMs, the model that performed the best had a test accuracy of 95.3%. It utilized a polynomial transformation kernel with a degree of 3 and regularization hyperparameter of 10. One thing worth mentioning is that the same kernel with regularization hyperparameters of 10 through 10^6 all had the same accuracies, suggesting that after performing a polynomial transformation, a more generalized line resulting from underfitting offered the best accuracy for SVMs. Among the transformations, no transformations had the worst performance with a large difference between the training and test accuracies. The other transformations had similar performances, though a polynomial transformation of degree three had the best performance.

Finally the hyperparameters that achieved the highest performance for the Neural Network model had a test accuracy of 0.958, though when tested on another test set, it dropped to an accuracy of 0.92. The hyperparameters included a ReLu activation function, a three level hidden layer architecture of the form [100,60,20], and a C value of 10. Looking at the regularization

hyperparameter, we can see that for very low C values, the training accuracies were very high, around 0.999 or 1. However, with these high training accuracies, we had notably lower training accuracies around 0.9, which were pretty low for our data set. This is likely due to the model overfitting to the training data, which resulted in high variance and fitting to noise, which affected how our model performed on the test data. On the other end of the spectrum, with high C values, we have high biases and the model under fits the training data. With high C values, both the training and testing accuracies plummet to a 0.43. Because we didn't perform feature transformations, the data did not have a more generalizable curve and had a different pattern in their accuracies vs C values graph. Among the different activation functions, they all performed with the same pattern of overfitting for low C values and underfitting for high ones, though their accuracies for ideal conditions only differed by trace amounts.

Looking at the confusion matrices for all three models, we can see a pattern, where the most common misclassifications involve the cat and sheep drawings. This could have been a result of the classes that were selected during classification. As mentioned earlier, the most common drawing of a cat involves its face, and the roundness of a cat's face matches that of a sheep's body. This discrepancy is particularly noticeable in the Neural Network model, where the True positive rate for sheeps and cats were 0.9 and 0.89 respectively whereas the giraffe drawings which had more contrast were correctly classified 96% of the time. From the same confusion matrix, we could see that the misclassifications were more frequently misclassifying a cat as a sheep or a sheep as a cat. Taking the workings of a Neural Network into consideration, this makes sense, as the similar features between the cat face and sheep could be identified as features in hidden layers, which may lead to the misclassification. A pattern also emerged among the penalty values. Higher C values worked best with SVM's and Logistic Regression, while lower values worked best with Neural Networks. This could be due to the fact that SVM and Logistic models are linear and would struggle to fit complex data with high regularization(As higher C values meant less regularization). However, as neural networks are better at fitting complex features in data, and since the data had high variance, the model performed better with lower C values.

Overall, the problem was solved successfully with all three models. Neural networks performed the best, but all models classified pictures reasonably well. Further improvements in the future could involve preprocessing the data by flipping, rotating, or scaling the image to add more data, as well as optimizing memory usage to be able to train with more examples, both of which could likely lead to higher accuracies.

Works Cited

Google Quickdraw Dataset
https://quickdraw.withgoogle.com/data

https://github.com/googlecreativelab/quickdraw-dataset

sklearn logistic regression documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

sklearn SVM documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

sklearn neural network documentation:
https://www.pluralsight.com/guides/machine-learning-neural-networks-scikit-learn

sklearn Logistic regression documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html