1. Introduction & Background Information

In 2016, Google released a game called “Quick, Draw!”[1]. The point of the game is simple, it gives the player a word and 20 seconds to draw that word. The pretrained model constantly updates its guess about the drawing. The game generated one billion drawings and from them, a unique open-source dataset, made of 50 million drawings, was created. In this project we implemented several ML models and used them for the task of classifying 13 animals from the shared dataset. We aimed to give the correct label for an animal drawing given by a user.

We performed experiments using different models. We first trained some models using Machine Learning algorithms like knn, SVC, boosting, logistic regression and random forest. Then we trained CNN models using the architecture proposed by the authors of [2]. Finally we used a transfer learning approach to get a pretrained network on images and use it for our cause. The network was ResNET50 [4].

We performed experiments for 3 classes (bear, giraffe, octopus) and 13 classes (bear, bird, dog, camel, dolphin,snake,snail,swan,zebra,cat,giraffe,octopus and fish) separately. From the experiments we performed using just 3 classes we observed that SVC showed 0.90 test accuracy while CNN and ResNet having test accuracy of 0.95 and 0.97 respectively. For the experiments performed using 13 classes, most of the ML algorithms performed poorly, SVC having the best score with 0.63 while Lenet and Resnet50 showing an accuracy of 0,86 and 0,87 respectively.

2. Problem Description

Analyzing and processing the visual imagery is a problem whose importance has increased over the years. Today, we have algorithms specifically designed and trained to perform this job. These models even take their place in our phones. The most widely used models for the area is Convolutional Neural Networks. So we selected this task to have hands-on experience on Convolutional NEtworks and image processing. Our problem is to label a drawing of an animal, given by a user via an app or the web browser. We solved this problem using the model we trained on the drawings of the selected 13 animals in the dataset. Our task is just like the Quick Draw game on a small scale. Instead of 345 categories we have just 13 categories to label which are namely; bear, bird, dog, camel, dolphin,snake,snail,swan,zebra,cat,giraffe,octopus and fish .

3. Dataset

There are 50 million drawings in the dataset which are collected by the help of 15 million players. In total there are 345 categories available, but we didn’t use all of them due to the dataset’s size, and required computational resources. We instead used 3 and 13 animals’ drawings for different test scenarios. The dataset can be used in either raw or simplified format. Raw data holds more information regarding the drawings. It has six columns:

- Key_id (integer): Holds a unique value that identifies the drawing.
- Word (string): Category of the drawing (what the player was asked to draw). This can be multiple words, but the spaces will be replaced with underscores.
- Recognized (boolean): Whether the AI guessed what the drawing was.
- Timestamp (datetime): When the drawing was created.
- Countrycode (string): A two letter code referring to the player's country.
- Drawing (string): Vector drawing in array form.

The drawing array is made of strokes. So, it holds pixel coordinates and the time for those coordinates. For a better understanding, a value from the drawing column would happen to look like this:

```
[
 [  // First stroke
   [x0, x1, x2, x3, ...],
   [y0, y1, y2, y3, ...],
   [t0, t1, t2, t3, ...]
 ],
 [  // Second stroke
   [x0, x1, x2, x3, ...],
   [y0, y1, y2, y3, ...],
   [t0, t1, t2, t3, ...]
 ],
 ... // Additional strokes]
```

In this array, x and y are the pixel coordinates, t is the time in milliseconds since the first point. x and y are real-valued, t is an integer. Drawing can differ in bounding boxes and number of points, since the devices used for display and input are different. In this way, a player's drawing can be simulated, hence, opening up more project ideas similar to Sketch-RNN. But for our project we only need the still image of a drawing to use it for classification. For this purpose simplified data is more fitting.

There are different file types available with different vector adjustments; simplified drawing files (.ndjson), binary files (.bin), numpy bitmaps (.npy). In the simplified version, time information is removed, data is positioned and scaled into a 256x256 region. For our research, we used numpy bitmaps, in which the drawings are rendered into a 28x28 grayscale bitmap. Three 28x28 grayscale drawing examples from the camel class can be seen in figure 1.

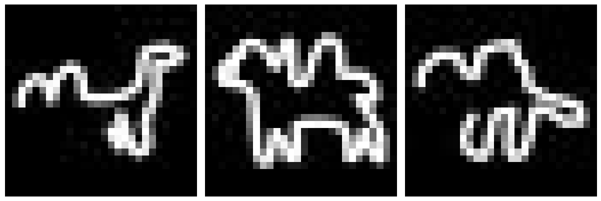


Figure1: Example drawings from the dataset

4. Methods

4.1. General Purpose ML Algorithms

Our aim was to implement different algorithms and choose the one with highest test performance to build the final model. We've first used ML algorithms which are not specifically designed to handle image data like, Adaptive Boosting, Bagging Classifier, Decision Trees, Random Forest, Gradient Boosting, Extreme Gradient Boosting, Logistic Regression, Bernoulli Naive Bayes, Gaussian Naive Bayes, KNN Classifier, Support Vector Machine and CNN.

To avoid overfitting, we set maximum depths for Random Forest, Decision Tree, XG Boosting and Gradient Boosting as 3 and train initial models. To compare the machine learning model we used following performance metrics: accuracy, precision and recall.

We fine tuned the parameters of KNN and SVC. We applied a grid search algorithm to find the hyperparameters that maximizes the test accuracy for KNN and SVM. We tried [3,5,10,15, 20, 30] for neighbor values for KNN and the n value that maximizes the test accuracy is 10. For SVM we tried c = [0.1,1,10], gamma=[scale,1,0.1,0.01], kernel = rbf and the values that maximizes the performance are c=10, gamma=scale. After that we applied a voting classifier using soft voting for the best 5 of the models including the fine tuned ones.

4.2. CNN Architecture

We implemented the Lenet-5 as our CNN architecture[2] since it consists of just 7 layers and performs good on image data. It has 7 layers: 3 convolutional layers, 2 subsampling layers, and 2 fully connected layers. For the subsampling part we have tried both max pooling and average pooling. Max pooling algorithm gave the better results that's why we decided to use it in our subsampling layers. The model can be seen in figure 2. After some trials we decided to add dropout with rate (0.2) between fully connected layers to prevent overfitting and increase test case accuracy

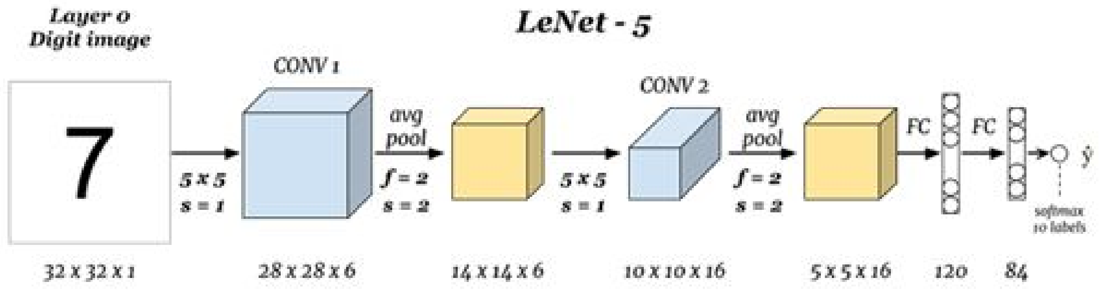


Figure 2: Lenet-5 Architecture

In the first convolutional layer there are 6 filters of size (5,5). Then max pooling applied with kernel size (2,2) to subsample. In the second convolutional layer, 16 filters with size (5,5) used. A max pooling layer with kernel size (2,2) follows this layer. After that we have 2 fully connected layers, the first one with 120 nodes and the second one with 84 nodes. At the end we have a softmax layer. The number of nodes in the output layer for 3-class prediction and 13-class prediction cases are 3 and 13respectively. For each layer, the number of parameters to be learned can be seen in the table 1. In total, we are learning 61691 parameters.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_2 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_2 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_3 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_4 (Dense)              (None, 13)                1105
=================================================================
Total params: 61,961
Trainable params: 61,961
Non-trainable params: 0
```

Table 1: Number of parameters for each layer of Lenet-5

4.3. Transfer Learning (ResNet50)

We applied transfer learning using ResNet50. It consists of 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. The Resnet50 model consists of 5 stages each with a convolution and identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers[3]. The model has 25,636,712 parameters.

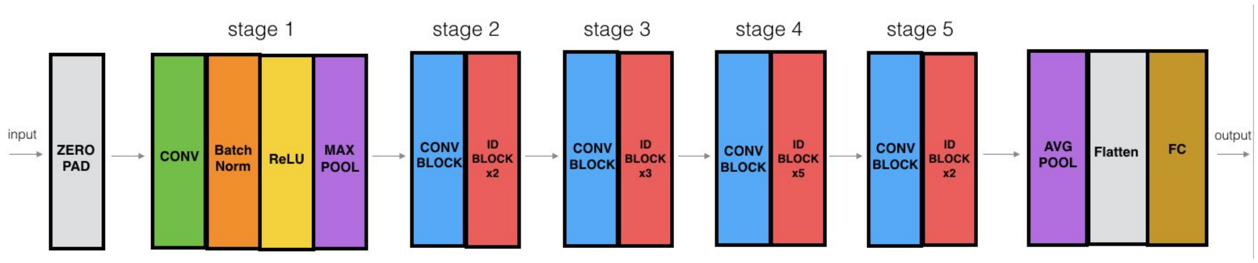


Figure 3: Resnet50 Architecture

When we choose weights=None, it initializes weights as random. To get better results we set weights as 'imagenet' to load the pretrained ImageNet weights which is found using We set include_top as false to not include a fully connected layer in the original model. Instead of using the fully connected network in the Resnet50 architecture we added 3 dense layers. The input shape should have exactly 3 channels, and width and height should be larger than 32. Therefore, we added processed our images and converted their shape from 28x28x1 to 32x32x3 before giving them to Resnet50.

4.4 Data Preprocessing

To implement ML algorithms we converted 28x28 numpy bitmaps to 784x1 numpy arrays and 28x28 for CNN. We added a pre-processing layer to convert 28x28x1 images to 32x32x3 images for transfer learning since Resnet50 is a model designed for images with 3 channels . We did this process by replicating a single channel (since our data is grayscale, it has 1 channel) 3 times and padding the sphere of our input images with zeros to make them 32x32 pixels. Since we copied the grayscale layer into a set of three RGB layers with the same value in each, we didn't lose any colour information when conforming to what the transfer learning expects in terms of dimensions. Then, we divide the values in each pixel by 255 to normalize our data. As class labels are strings (eg. "octopus"), we converted them to categorical values with one-hot-encoding.

Data was splitted in a stratified fashion into train and test with 80% and 20% respectively. Class labels are equally distributed. We trained ML models with different sample sizes & categories. We changed the test size to 5% for scenario 6. Since we used 20000 samples for each class, 5% of them would be sufficient enough to see the performance of our model.

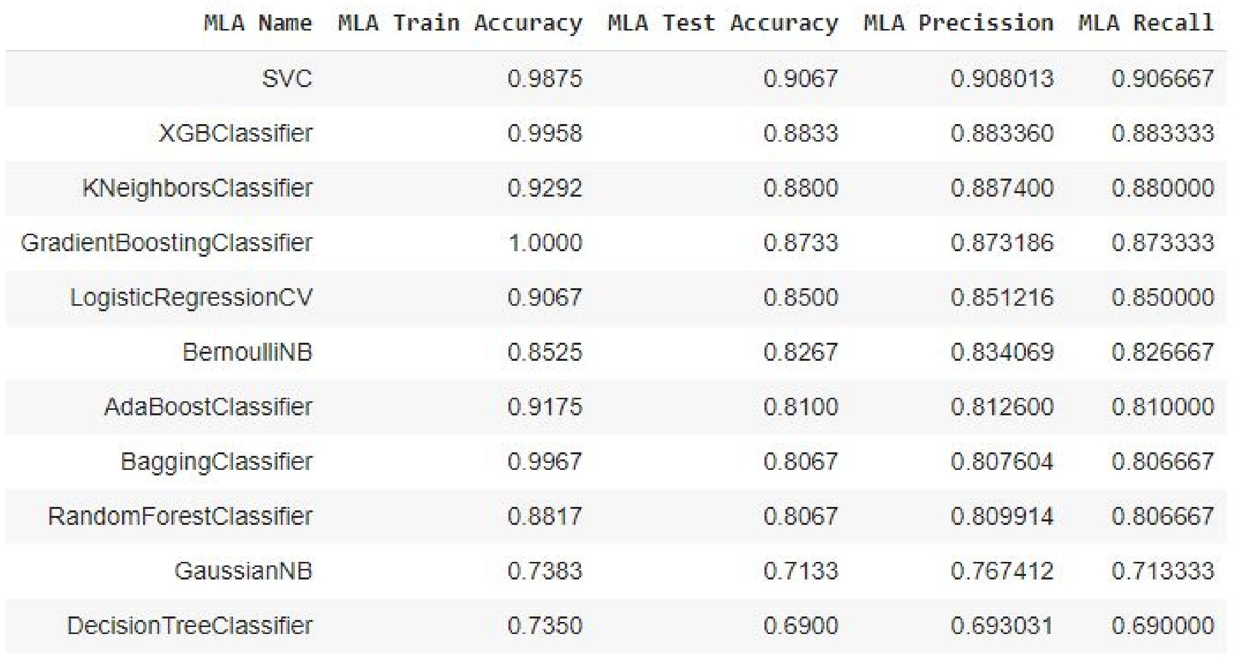
5. Results

We performed tests using different scenarios to find out the good and bad aspects of each model. Our models and experiments can be grouped into 3 categories which are general purpose ML algorithms, CNN algorithms, and transfer learning methods.

5.1. General Purpose Machine Learning Algorithms

5.1.1. Scenario 1:

We trained the aforementioned ML algorithms except CNN for 3 classes, using 500 samples for each class. In terms of test accuracy SVC gave us the best result with 79.3%. Its precision and recall were 0.791 and 0.793 respectively. The results can be seen on table 4.

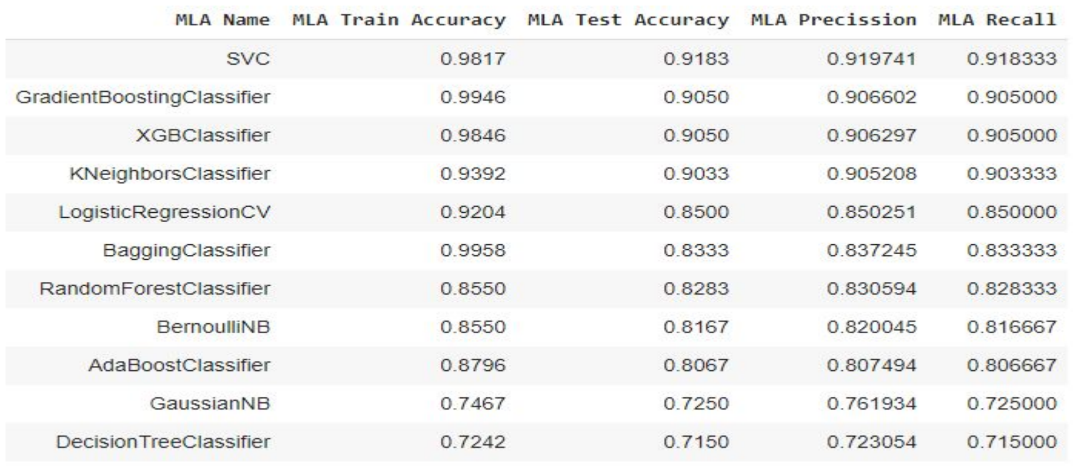


MLA Name	MLA Train Accuracy	MLA Test Accuracy	MLA Precission	MLA Recall
SVC	0.9875	0.9067	0.908013	0.906667
XGBClassifier	0.9958	0.8833	0.883360	0.883333
KNeighborsClassifier	0.9292	0.8800	0.887400	0.880000
GradientBoostingClassifier	1.0000	0.8733	0.873186	0.873333
LogisticRegressionCV	0.9067	0.8500	0.851216	0.850000
BernoulliNB	0.8525	0.8267	0.834069	0.826667
AdaBoostClassifier	0.9175	0.8100	0.812600	0.810000
BaggingClassifier	0.9967	0.8067	0.807604	0.806667
RandomForestClassifier	0.8817	0.8067	0.809914	0.806667
GaussianNB	0.7383	0.7133	0.767412	0.713333
DecisionTreeClassifier	0.7350	0.6900	0.693031	0.690000

Table 2: Results observed for scenario 1

5.1.2. Scenario 2:

We trained the aforementioned ML algorithms except CNN for 3 classes, using 1000 samples for each class. In terms of test accuracy GradientBoostingClassifier gave us the best result with 83.1% followed by SVC with 83%. Doubling the data size from 500 samples per class to 1000 per class, increased the best accuracy by 3%. The results can be seen in table 5.



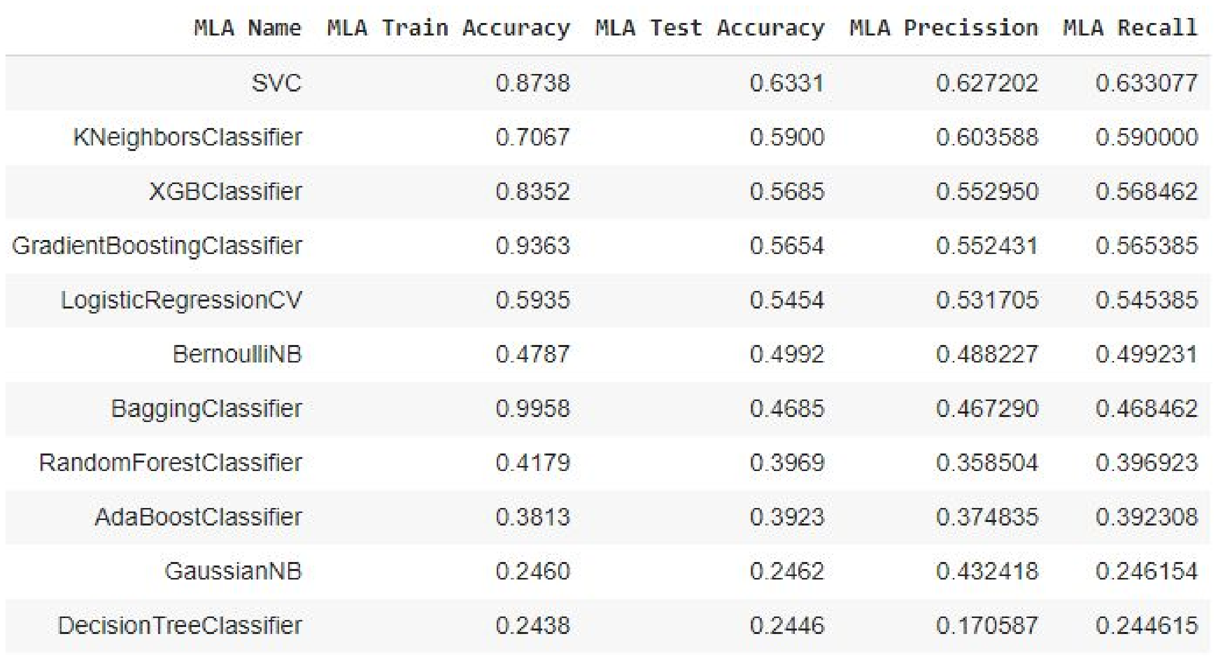
MLA Name	MLA Train Accuracy	MLA Test Accuracy	MLA Precission	MLA Recall
SVC	0.9817	0.9183	0.919741	0.918333
GradientBoostingClassifier	0.9946	0.9050	0.906602	0.905000
XGBClassifier	0.9846	0.9050	0.906297	0.905000
KNeighborsClassifier	0.9392	0.9033	0.905208	0.903333
LogisticRegressionCV	0.9204	0.8500	0.850251	0.850000
BaggingClassifier	0.9958	0.8333	0.837245	0.833333
RandomForestClassifier	0.8550	0.8283	0.830594	0.828333
BernoulliNB	0.8550	0.8167	0.820045	0.816667
AdaBoostClassifier	0.8796	0.8067	0.807494	0.806667
GaussianNB	0.7467	0.7250	0.761934	0.725000
DecisionTreeClassifier	0.7242	0.7150	0.723054	0.715000

Table 3: Results observed for scenario 2

After fine tuning the hyperparameters of KNN and SVC their accuracies increased by 1% and 2% respectively. Then we picked up the best 5 algorithms and applied voting classifiers with soft voting and observed an accuracy of 83%.

5.1.3. Scenario 3:

For Scenario 3, we used KNN and SVC with their fine tuned hyperparameters. We trained the aforementioned ML algorithms except CNN for 12 classes, using 1000 samples for each class. In terms of test accuracy SVC gave us the best result with 62.1% followed by KNN with 60.7%. Increasing the number of classes from 3 to 12 decreased the best accuracy by 21% as can be seen in the table 6.



MLA Name	MLA Train Accuracy	MLA Test Accuracy	MLA Precission	MLA Recall
SVC	0.8738	0.6331	0.627202	0.633077
KNeighborsClassifier	0.7067	0.5900	0.603588	0.590000
XGBClassifier	0.8352	0.5685	0.552950	0.568462
GradientBoostingClassifier	0.9363	0.5654	0.552431	0.565385
LogisticRegressionCV	0.5935	0.5454	0.531705	0.545385
BernoulliNB	0.4787	0.4992	0.488227	0.499231
BaggingClassifier	0.9958	0.4685	0.467290	0.468462
RandomForestClassifier	0.4179	0.3969	0.358504	0.396923
AdaBoostClassifier	0.3813	0.3923	0.374835	0.392308
GaussianNB	0.2460	0.2462	0.432418	0.246154
DecisionTreeClassifier	0.2438	0.2446	0.170587	0.244615

Table 5: Results observed for scenario 3

5.2. CNN Model

5.2.1. Scenario 1

We trained our CNN model (lenet-5) with 3 classes and 1000 samples for each. The observed accuracy was 88.1% which is 8.8% higher than the previous models.

5.2.2. Scenario 2

We trained our CNN model (lenet-5) with 12 classes and 1000 samples for each class. The observed accuracy was 83.5% which is 21.3% higher than the previous models.

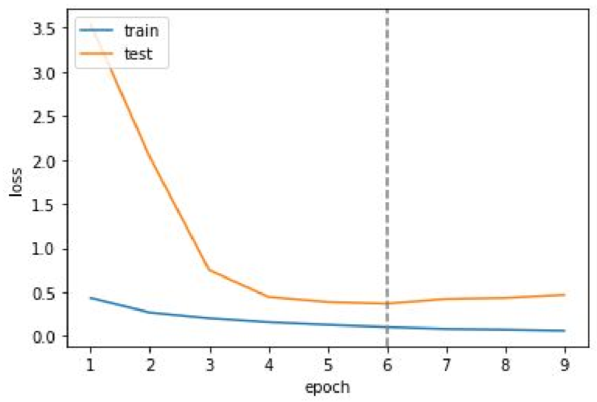
5.2.3. Scenario 3

We trained our CNN model (lenet-5) with 12 classes and 20000 samples for each class. The observed accuracy was 84.1% which is 0.6% higher than the model with 1000 samples.

5.3. Transfer Learning

5.3.1. Scenario 1

We implemented a pretrained model Resnet50 with 3 classes and 20000 samples in each class. We used Adam optimizer with 0.0001 learning rate and categorical cross entropy loss function. To determine the number of epochs, we applied early stopping and chose the iteration with minimum validation accuracy. The resulting train and test accuracies are 97.6% and 89.2% respectively.



5.3.2. Scenario 2

We applied transfer learning, Resnet50 architecture with 13 classes and 20000 samples in each class. The observed training accuracy was 92.42 % and the test accuracy was 87.09 %

Below table summarizes the implemented models and their results:

Algorithms	# Classes	Sample Size per Class	Best Test Accuracy
General Purpose ML	3	500	90.67%
General Purpose ML	3	1000	91.83%
General Purpose ML	13	500	63.31%
CNN Lenet-5	3	500	92.33%
CNN Lenet-5	13	1000	80.42%
CNN Lenet-5	13	20000	87.05%
Transfer Learning Resnet50	3	20000	89.21%
Transfer Learning Resnet50	13	20000	87.09%

Table 6: Best test accuracies observed in each scenario

6. Discussion

Results for the general purpose ML algorithms show that for a small number of classes(3) their performance is good, having a test accuracy of 0.90 even with a smaller number of data(500). Results indicate the performance improves as we increase the number of training data from 500 to 1000 as well for 3 classes. For 13 classes they performed badly, best one having a test accuracy of 0.63. Increasing the number of training data does not improve performance for that case and it comes with its own problems. Since general purpose algorithms don't take advantage of parallelism much, the time required for training the models increases a lot. Therefore, it is better to use algorithms which take advantage of the parallelism available while dealing with images.

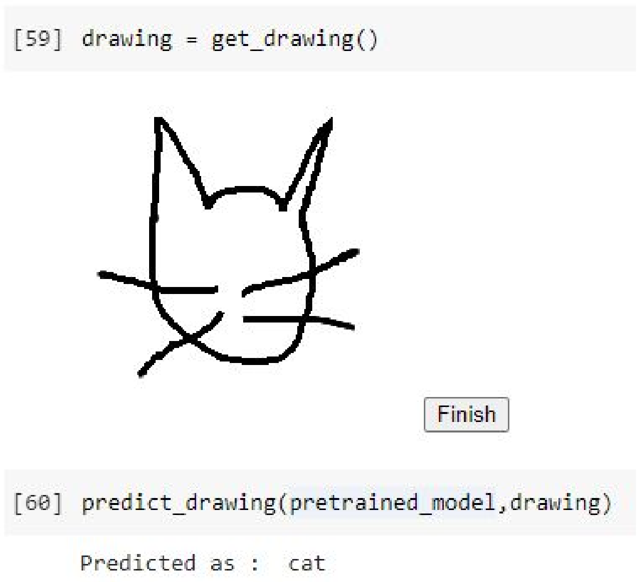
The result for the CNN architecture that we implemented clearly shows that they are very successful at dealing with images even with small numbers of data having an accuracy of 92.33 even with 500 samples from each class. For 13 classes they also perform 17% better than the general purpose ML algorithms. Since they took advantage of parallelism, it was also possible to train them with large numbers of data(20000 samples per class). Increasing the number of samples, resulted in an accuracy increase of 7%.

The pretrained models are also useful, but it required us to make the necessary shape changes in the input before giving to the pretrained model in our case ResNet50. The performance even boosted by 2.2% when we used the Resnet for 3 classes. For 13 classes it performed slightly better than the precious CNN architecture. We normally expected it to perform much better than our CNN model. We think that this may be caused by the changes that we made to our data before giving it to ResNEt50. As we mentioned, our data has 1 channel but Resnet trained on images with 3 channels. Also, since our images are easier to understand then real world images, using ResNet50 may be an overkill for this purpose.

7. Conclusion

Although some of general machine learning algorithms perform as good as CNN with smaller class sizes, the performance drops rapidly then the class size is increased. Therefore, we applied transfer learning with ResNet50 architecture that worked best with 13 classes of animals.

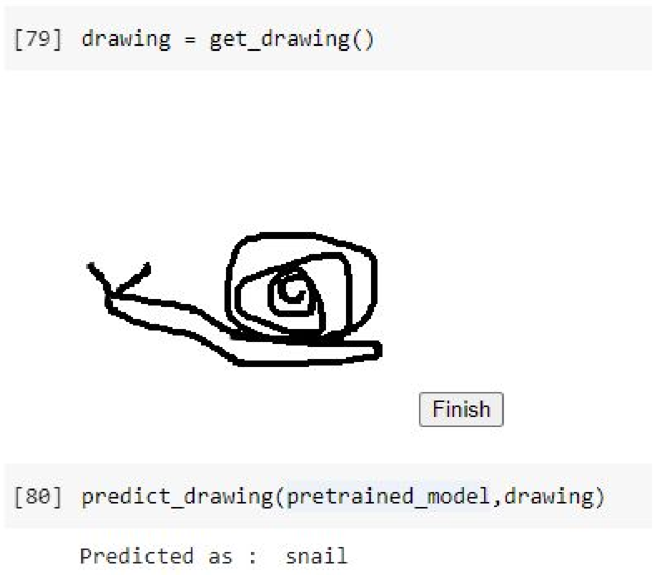
Then, we added an interactive painting box to the notebook to test the model.



```
[59] drawing = get_drawing()
```

```
[60] predict_drawing(pretrained_model,drawing)

    Predicted as :  cat
```



```
[79] drawing = get_drawing()
```

```
[80] predict_drawing(pretrained_model,drawing)

    Predicted as :  snail
```

As future interests, robust optimization can be used instead of Adam optimizer to deal with the noise of the data. We can generate data to make our model faster and memory efficient. Additionally, we can add country information that the drawing is drawn to our input data since different countries draw different things in different ways.

8. Division of Work

Büşra Erdal	Abdullah Furkan Okuyucu	Elif Naz Özdamar	Elif Şahin
Report Preparing Presentation Preparing data for processing Model Selection Model Training Obtaining results Transfer Learning	Report Preparing data for processing Model Selection Model Training Implementation of CNN Hyperparameter Tuning Transfer Learning Getting User Input	Report Preparing Presentation Model Selection Model Training Obtaining Results Hyperparameter Tuning Transfer learning	Report Preparing data for processing

References

1. Experiments.withgoogle.com. 2020. *Quick, Draw! By Google Creative Lab | Experiments With Google*. [online] Available at: <https://experiments.withgoogle.com/quick-draw> [Accessed 16 December 2020].
2. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.
3. Dwivedi, Priya. “Understanding and Coding a ResNet in Keras.” *Medium*, Towards Data Science, 27 Mar. 2019, towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33.
4. K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

```
[59] drawing = get_drawing()
```

Finish

```
[60] predict_drawing(pretrained_model,drawing)

     Predicted as :  cat
```

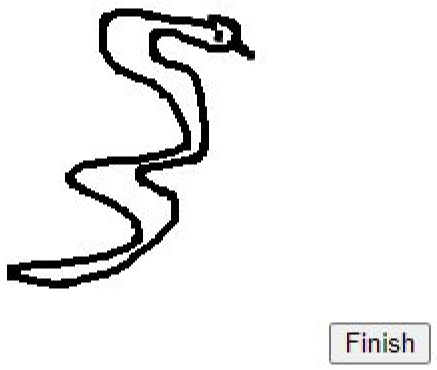
```
[65] drawing = get_drawing()
```

Finish

```
[66] predict_drawing(pretrained_model,drawing)

     Predicted as :  bear
```

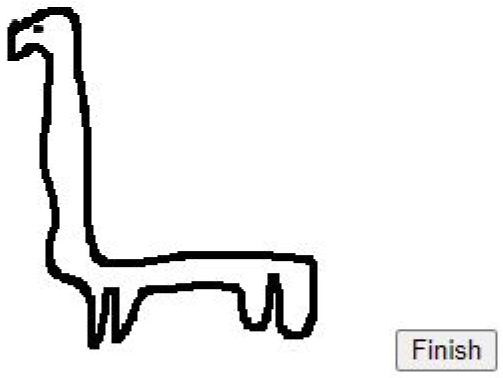
```
[67] drawing = get_drawing()
```



```
[68] predict_drawing(pretrained_model,drawing)
```

```
Predicted as :  snake
```

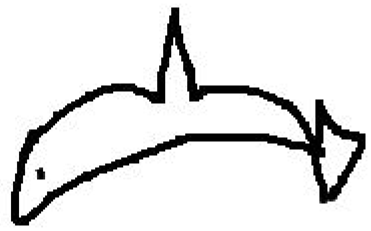
```
[69] drawing = get_drawing()
```



```
[70] predict_drawing(pretrained_model,drawing)
```

```
Predicted as :  giraffe
```

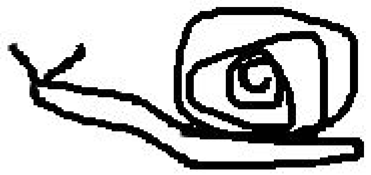
```
drawing = get_drawing()
```



```
[76] predict_drawing(pretrained_model,drawing)
```

```
Predicted as :  dolphin
```

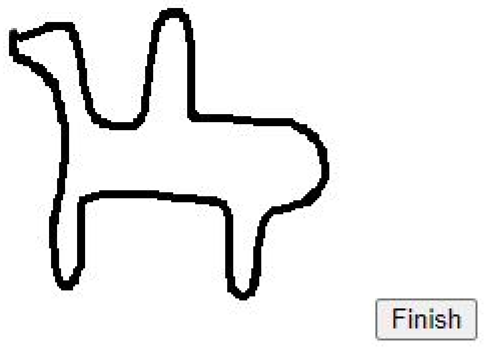
```
[79] drawing = get_drawing()
```



```
[80] predict_drawing(pretrained_model,drawing)
```

```
Predicted as :  snail
```

```
[81]  drawing = get_drawing()
```



```
[82]  predict_drawing(pretrained_model,drawing)

      Predicted as :   camel
```

```
[84]  drawing = get_drawing()
```

```
[85]  predict_drawing(pretrained_model,drawing)

      Predicted as :   octopus
```