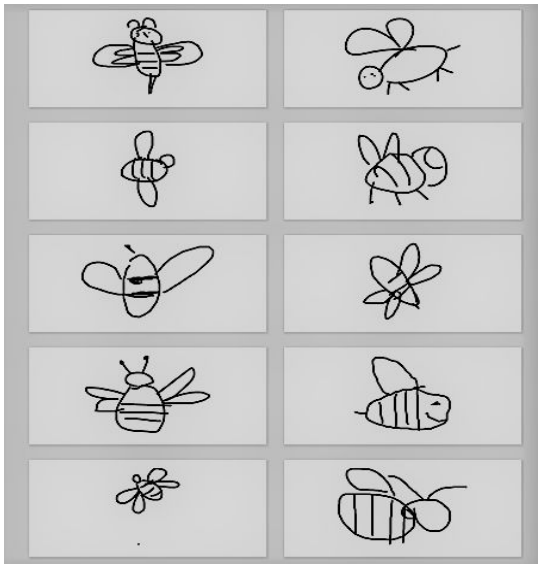
Quick, Draw! Doodle Challenge

Learning Unstructured Features for Doodle Detection

Aravind Kandiah | Charles Wong | Keshigeyan Chandrasegaran
Singapore University of Technology and Design (SUTD)

Abstract
Quick, Draw was released as an experimental game to educate the public in a playful way about how Artificial Intelligence (AI) works. The game prompts users to draw an image depicting certain a category such as ‘car’, ‘suitcase’ etc. The game generated more than 1 billion drawings where a subset depicting 50M drawings with 340 categories were publicly made available to enable building a better classifier.

This is a hard computational problem since it directly involves learning unstructured features from noisy data for object classification. i.e.: There are more than 10 ways of drawing a bee and the learning algorithm should ideally be able to learn the inherent features of the bee using the noisy data provided by the users.



Introduction
In this paper, we qualitatively and quantitatively postulate our hypotheses, describe the different methods we have utilized and present each of their corresponding results.

The following gives an overview of the list of methods we have utilized

- Convolutional Neural Networks
- Mobilenets
- ResNets
- Filtering using Image Entropy

Data
The raw data is the exact input recorded from the user drawing and stored as follows:

Key	*Type*	*Description*
key_id	64-bit uint	A unique identifier across all drawings.
word	string	Category the player was prompted to draw.
recognized	boolean	Whether the word was recognized by the game.
timestamp	datetime	When the drawing was created.
countrycode	string	A two letter country code of where the player was located.
drawing	string	A JSON array representing the vector drawing

The information about the drawing is stored as a list of list of strokes of the drawing. Below is the format of a drawing vector.

```
{
        [  // 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
        ]
}
```

x and y are the pixel coordinates, and t is the time in milliseconds since the first point. x and y are real-valued while t is an integer.

Data Reduction

Initially, the dataset of 50 million points took up 73GB of space. In order to make computation more efficient, the dataset was simplified. Simplification was done by 2 steps. First, by simplifying line segments, straight line segments that were initially represented by 15 points are represented by 2 points. So we can now represent the same image with much fewer coordinates. The time stamps for each of the points are also discarded while preserving the order of the strokes. So the temporality of the drawing is still maintained but with lower fidelity.

Data Generator

Given the stroke information, a generator was created to render the drawings and serve as the input form for the model

Temporality Feature Inclusion

As the data has temporal information in its data, we wanted to represent the temporal information together with the image. In order to do so we gradually changed the shade of each stroke after the 10th stroke, so each subsequent stroke after the 10th stroke would get lighter and lighter.

We hypothesize that this may help in two ways. Firstly be emphasizing the initial phase of the drawing, as the model may learn to give more importance to the abstract and overall shapes and secondly it may give less importance to the more micro features and details that people tend to draw later on in the drawing. As these micro features and details can really add significant noise to the image and vary greatly from person to person.

Data Exploration

Due to the crowdsourced nature of how the data was collected, it has resulted in a very noisy dataset. Mentioned below are all the possible resulting data types:

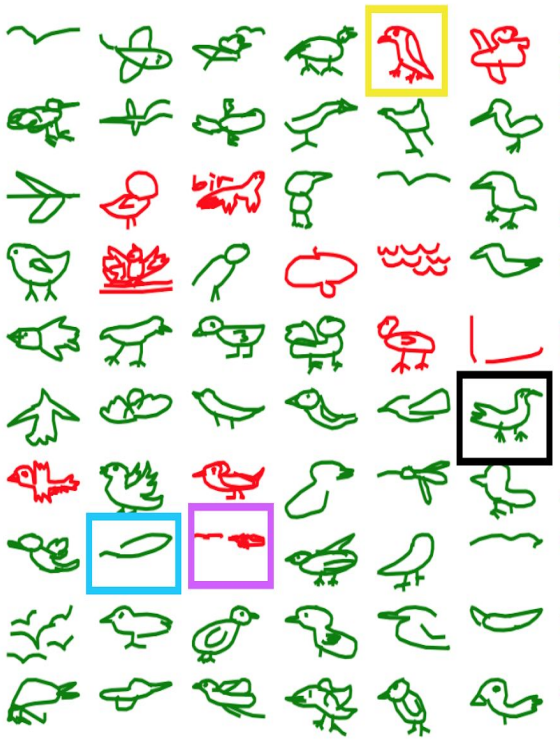
1. True positive: Drawer draws a correct doodle, and Google marks it as ‘successful recognition’.
2. True negative: Drawer draws a wrong doodle, and Google marks it as ‘failed recognition’.

3. False positive: Drawer draws a wrong doodle, but Google marks it as ‘successful recognition’.

4. False negative: Drawer draws a correct doodle, but Google marks it as ‘failed recognition’.

The noisy labeled data is introduced by 2 and 3, but if we delete all ‘failed recognition’ data, we end up deleting 2 and 4.

Recognized Label (Green)
Not Recognised Label (Red)
True Positive (Black Box)
True Negative (Purple Box)
False Positive (Blue Box)
False Negative (Yellow Box)



From our data exploration, we also noticed that there were a considerable amount of unrecognized images where users tried to cancel their drawings by scribbling or their drawing is too messy to be even deciphered by humans.

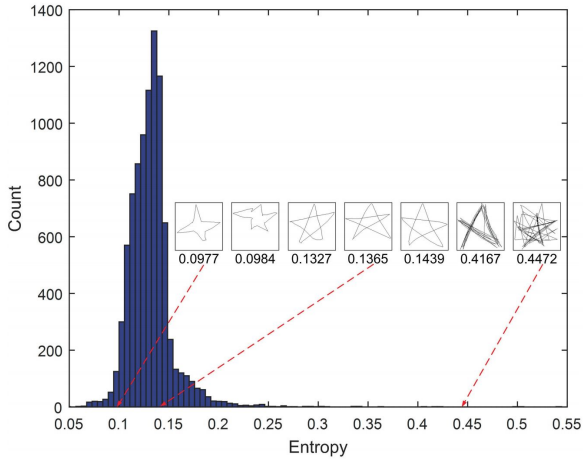
Example of such cases

Filtering Out Noisy Labels

In order to remove such images with an abnormal amount of strokes, we devised the following method to preprocess the data inspired by the paper on “SketchMate: Deep Hashing for Million-Scale Human Sketch Retrieval”.

http://openaccess.thecvf.com/content_cvpr_2018/CameraReady/2763.pdf

For each class we calculate the Shannon image entropy for each sketch and the overall entropy distribution on a category basis.



Based on empirical tests and visual inspection of the results we found that removing the 0.003 and 0.985 percentile tail-end outliers in the distribution, resulted in the best results. As low entropy (0.003) doodles were often overly abstract, often wrong and do not convey much detail about the class. While high entropy (0.985) drawings were extremely messy and often times just scribbles that are not discernable.

In order to introduce make this boundary values a hyperparameter we can vary at runtime. We normalized the entropy values by the distribution of each class and append it to the training file. This allowed us to use fixed z-values across varying classes to select our upper and lower boundary values.

Exploring Temporality and Shading

We tested out 2 different types of shading generators. The first generator rendered images that started at a value of 255 and turned lighter for subsequent strokes until it reached the 10th stroke, where it remained the same value for the remaining strokes. The second generator starts at 255 for the first 10 strokes and then decays to zero over the next subsequent strokes. The first generator preserves all the strokes while including temporality features in the first 10 strokes, the second model does not include temporality for the first 10 strokes but includes temporality in the strokes after 10 and decays to zero. The second generator is intentionally crafted such that it eventually fades unnecessary strokes at the end and presents the more abstract shapes that the initial strokes provide. Through our testing we found the second generator to perform better.

An example of the first generator is shown on the left and the second generator on the right.

Evaluation Metric

$$Map@3 = (1/U) \sum_{u=1}^{U} \sum_{k=1}^{min(n,3)} P(k)$$

U is the number of scored drawings in test data, P(k) is the precision at cutoff k and n is the number of predictions per drawing.

$$P(k) = number\ of\ correct\ predictions\ /\ total\ predictions$$

Train, Validation and Test Sets
For the initial series of experiments, we will randomly sample 1 million drawings from the 50M drawings where we will use 80% of the drawings for training (800k) and 20% of the drawings for validation. (200k)

We also have a separate 100k drawings test set in which we will evaluate the performance of different models after every experiment.

After every experiment, we will submit our predictions on the test set provided by Kaggle and report our Kaggle leaderboard score as well.

So effectively, we will have 2 MAP@3 scores for every experiment.

We have 3 Big Data experiments done on 17 million drawings for specific models and architectures. They are described under corresponding experiments clearly.

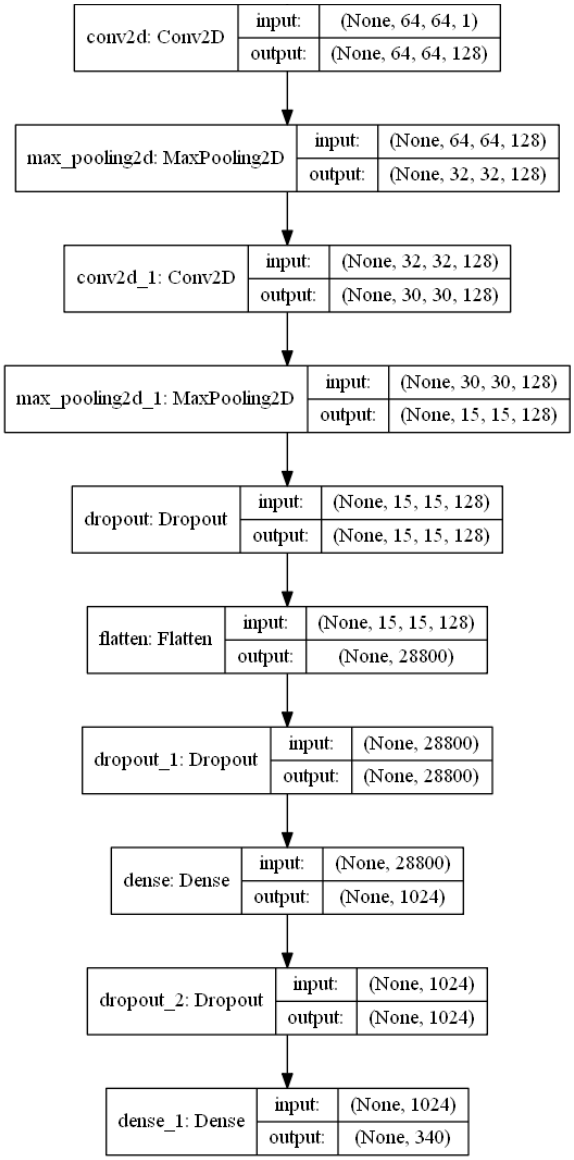
1. Convolutional Neural Networks

Convolutional Neural Networks are classical neural networks that serve as a good starting point for visual imagery analysis and prediction. So we will start our experiments and analysis with simple CNN architectures.

a. Experiment 1

Hypothesis: The stroke information is independent of the object representation. Essentially, a good classifier can be constructed by only learning the features of the object, ignoring how it is drawn.

CNN Architecture:



Total params: 29,989,588
Trainable params: 29,989,588
Non-trainable params: 0

Hyperparameters:
Image Input size = 64x64
Steps per Epoch = 1000
Batch size = 256
Epochs = 32
Initial Learning Rate = 0.001

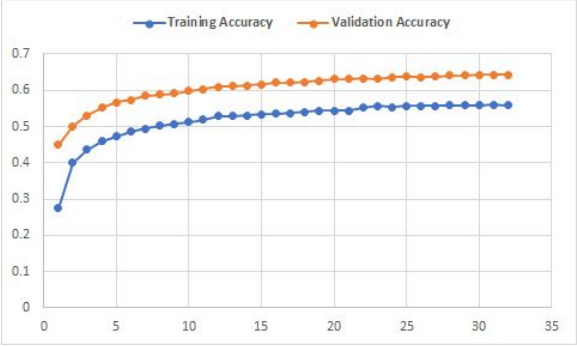
Reasoning for architecture and hyperparameters selection

Since we are undertaking our first experiment, we wanted to stick to a simple vanilla CNN architecture with 2 convolution layers. We also arrived at this architecture and hyperparameters after doing some preliminary analysis of publicly available kernels on Kaggle.

The primary purpose of a simple architecture is to get baseline performance metrics that can be improved on our subsequent iterations.

Computational resources and time are also constrained.

Performance
Map3 score on test set : 0.720
Kaggle Leaderboard Score : 0.581



Understanding the fit of data
Validation Accuracy is better than training accuracy. Validation loss is also lower than the training loss. This seems like an unknown fit to the data. We infer that this is caused by the dense dropout and convolution dropout layers which have 0.4 values in this architecture. In the following experiments, we will try to reduce this value by half to allow a better fit on the training data.

Inferences / Conclusion
Input data could be noisy. Feeding larger batch size could potentially result in more improvement. Stroke information is unlikely to be independent of the shape. Encoding stroke information is likely to provide better prediction.

b. Experiment 2

Hypothesis: Based on the inferences from experiment 1, feeding a larger batch size will result in better prediction.

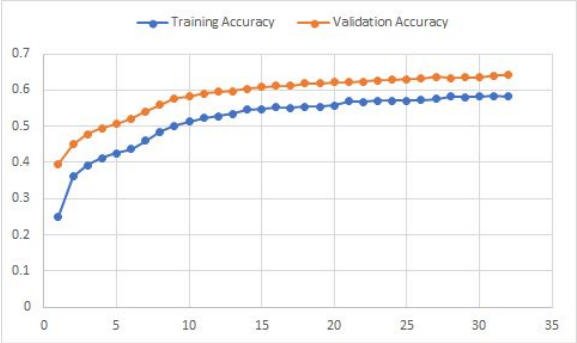
CNN Architecture: Identical architecture as experiment 1. The only change is to reduce the convolutional and dense dropout values from 40% (in experiment 1) to 20%.

Hyperparameters:
Image Input size = 64x64
Steps per Epoch = 800
Batch size = 512
Epochs = 32
Initial Learning Rate = 0.002

Reasoning for architecture and hyperparameters selection
Learning rate was increased from 0.001 to 0.002 to allow faster convergence. Also, step size was reduced to 800 due to computational resource constraints. Reducing the convolutional and dense

dropout values from 40% to 20% is to allow for a better fit on the training data.

Performance
Map3 score on test set : 0.717
Kaggle Leaderboard Score : 0.616



Understanding the fit of data
Still validation accuracy is better than training accuracy throughout. Validation loss is also lower than the training loss throughout. But the gap has significantly reduced suggesting that the dropout values are regularizing the system too much so that the training process generalizes very well on the validation set.

Inferences / Conclusion
This experiment provides same performance in predictions on our test data and provides 3.5% improvement in kaggle test set.

Some possible explanations for this observation could be (i) kaggle test set having sketches that are difficult to predict (2) our own test set having sketches that can be easily predicted. Hence Kaggle test set performance will be a better indicator of performance.

Thus feeding larger batch size of images do provide better predictions. A possible reason for this would be large amount of noisy data points, so a larger batch size can allow the network to learn correct features.

c. Experiment 3

Hypothesis: Encoding stroke information in the images will result in a better classifier.

CNN Architecture: Identical architecture as experiment 2. We will not change the dropout parameters in this experiment since it can introduce large variations. So we will leave it as a control variable for this experiment.

We will introduce an interesting approach to encode the time dependency of strokes in the images. We will darken the strokes as time progresses so that strokes initially drawn have more weightage (convey more information) compared to the subsequent strokes allowing the network to be able to understand better on how the objects are drawn.

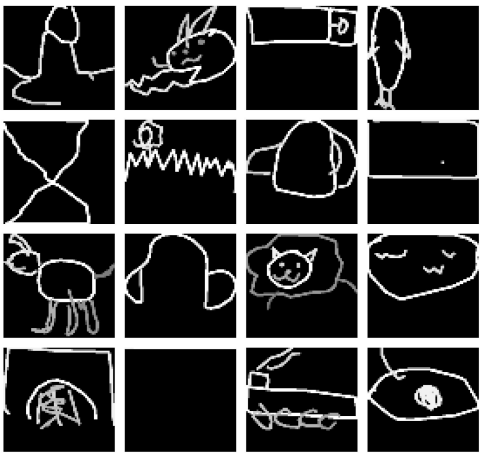
Hyperparameters:
Image Input size = 64x64
Steps per Epoch = 800
Batch size = 512
Epochs = 32

Initial Learning Rate = 0.002
Subsequent strokes shade reduction = 13
Number of strokes considered = 10

Reasoning for architecture and hyperparameters selection

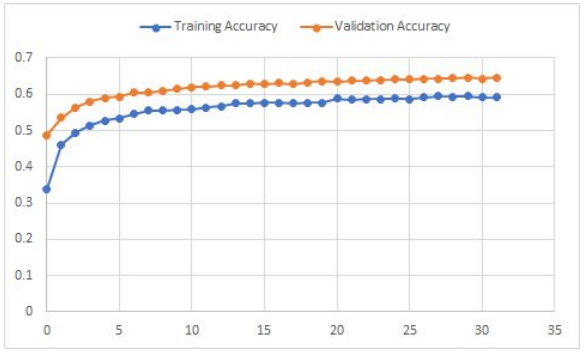
Learning rate was increased from 0.001 to 0.002 to allow faster convergence. Also step size was reduced to 800 due to computational resource constraints.

Number of strokes and strokes reduction value was decided after doing some minor experiments on encoding the stroke information as well as preserving most of the information. Refer to the diagram below for some sample sketches from the training data that have stroke information encoded.



Performance

Map3 score on test set : 0.722
Kaggle Leaderboard Score : 0.722



Understanding the fit of data

Performance on validation set is better than the training set since it is a control variable. But an important observation is that the overall fit of the data has improved vastly.

Inferences / Conclusion

This experiment provides same performance in predictions on our test data compared to experiment 3, but provides 10.6% improvement in predictions on the kaggle test set.

Thus encoding stroke information provides huge improvement in predictions.

2. MobileNets

Since typical convolutional networks are computationally heavy, we will use a different class of convolutional neural

networks that are efficient for mobile and embedded vision applications called mobileNets.

MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks.

a. Experiment 4

Hypothesis: Deeper networks explain the data well and provide better image predictions.

MobileNet Architecture: We are using the standard implementation of MobileNet. The paper can be found at, https://arxiv.org/pdf/1801.04381.pdf

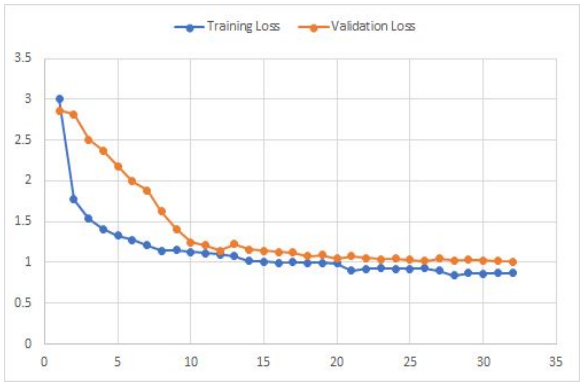
Total params: 3,576,788
Trainable params: 3,554,900
Non-trainable params: 21,888

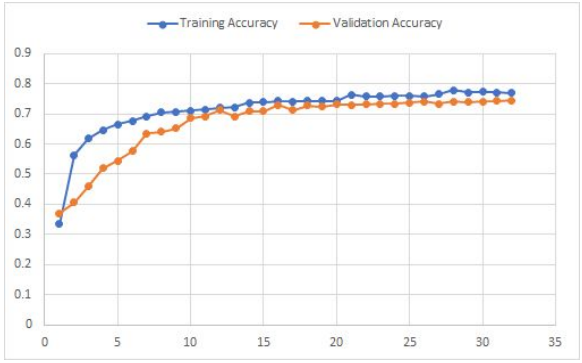
Hyperparameters:
Image Input size = 64x64
Steps per Epoch = 800
Batch size = 512
Epochs = 32
Initial Learning Rate = 0.002
Subsequent strokes shade reduction = 13
Number of strokes considered = 10

Reasoning for architecture and hyperparameters selection
We are using the standard implementation of MobileNet in Keras for this experiment.

The hyperparameters were not changed from the previous experiment. Since the architecture has completely changed, we wanted to keep the hyperparameters as control variables for this experiment.

Performance
Map3 score on test set : 0.810
Kaggle Leaderboard Score : 0.844





Understanding the fit of data
The experiment performs good fit on the data.

Inferences / Conclusion
This experiment provides 8.8% improvement in predictions on our test data and 12.2% improvement in predictions on the kaggle test set.

Deeper networks do provide better predictions and we plan to explore more fine-tuned versions of mobile nets for our future experiments.

b. Experiment 5

Hypothesis: Feeding in larger image size can allow for better predictions.

MobileNet Architecture: Same architecture as experiment 4. Only change is larger image size input.

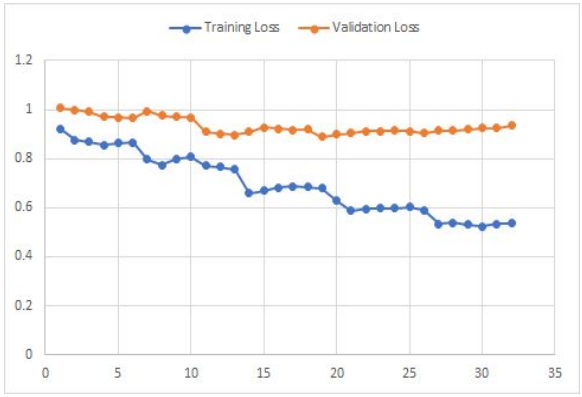
Hyperparameters:
Image Input size = 92x92
Steps per Epoch = 800
Batch size = 530
Epochs = 32
Initial Learning Rate = 0.002
Subsequent strokes shade reduction = 13
Number of strokes considered = 10

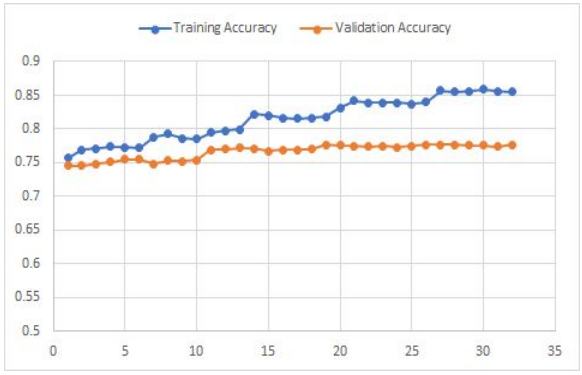
Reasoning for architecture and hyperparameters selection
The original images in the dataset were rendered to a 256x256 region. We downsampled the images to 64x64 in all our previous experiments due to computational constraints.

In this experiment we will downsample the images to 92x92, a larger size which can potentially allow the network to learn image features better.

We will also feed a larger batch size of 530 images per step to allow the network to learn better since there are many noisy data points (Inference from experiment 2). All the other hyperparameters are identical to the previous experiment.

Performance
Map3 score on test set : 0.836
Kaggle Leaderboard Score : 0.883





Understanding the fit of data
The experiment provides good fit on the data.

Inferences / Conclusion
This experiment provides 2.6% improvement in predictions on our test data and 3.9% improvement on the kaggle test set.

Thus feeding a larger size image allows the network to learn features better.

c. Experiment 6 (Big Data)

Hypothesis: Training on larger data provides better predictions at the expense of diminishing returns.

Data: This is the first big data experiment that we will carry out. In this experiment we will use 17 million images for the experiment where we will use 98% for training and 2% for validation sets.

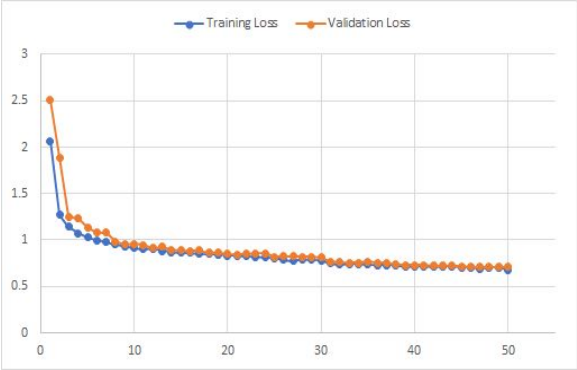
MobileNet Architecture: Same architecture as experiment 5.

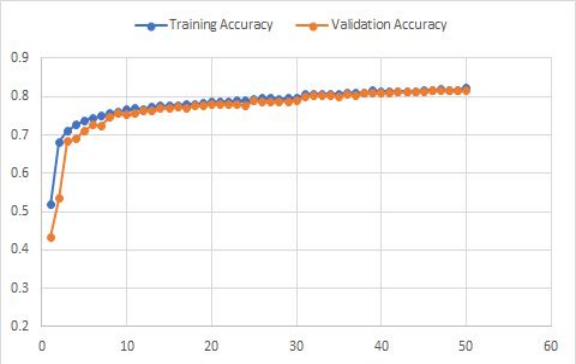
Hyperparameters:
Image Input size = 128x128
Steps per Epoch = 1000
Batch size = 680
Epochs = 50
Initial Learning Rate = 0.002
Subsequent strokes shade reduction = 13
Number of strokes considered = 10

Reasoning for architecture and hyperparameters selection
In this experiment we will downsample the images to 128x128, an even larger size compared to experiment 5 to allow the network to learn better.

Batch size will be increased to 680.
Epochs will be increased to 50 to train on the whole data.

Performance
Map3 score on test set : 0.867
Kaggle Leaderboard Score : 0.906



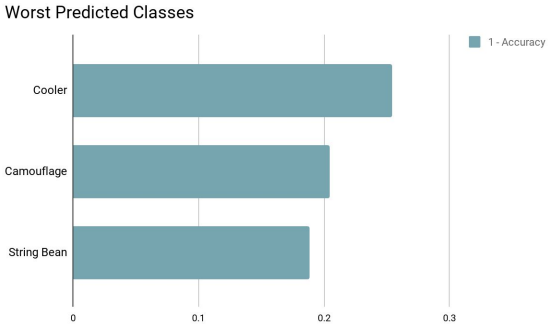


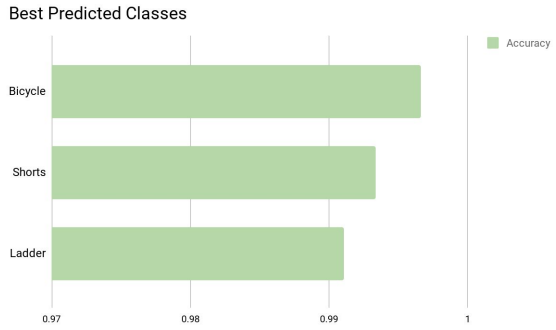
Understanding the fit of data
The experiment provides very good fit on the data.

Inferences / Conclusion
This experiment provides 5.7% improvement in predictions on our test data and 2.6% improvement on the kaggle test set.

This a great improvement. Thus we have sufficient evidence to accept our hypothesis.

Further exploration of data
We will perform more in-depth data exploration to understand which classes have higher and lower accuracies in our test set predictions.





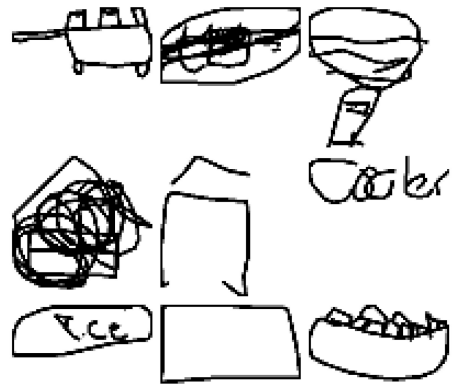
Cooler, Camouflage and String Bean class objects are very hard for the classifier to predict. We will explore some samples of these class objects and see how they are drawn.

The following are some sketches corresponding to Cooler class that the classifier was not able to identify.

The following are some sketches corresponding to Camouflage class that the classifier was not able to identify.

The following are some sketches corresponding to String Bean class that the classifier was not able to identify.

To understand this problem better, we checked some samples of images corresponding to Coolers, Camouflage and String beans in the training data and discovered large amount of noise in these class samples. Some of the noisy sketches in the train data for these class are shown below



Therefore, after model and architecture selection we will utilize stroke count based image processing technique described in the data exploration section to remove noisy data to train a better classifier.

3. Residual Networks

In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, the network learns residuals.

Residual can be simply understood as subtraction of feature learned from input of that layer. Residual networks achieve this by using shortcut connections from layer i to the (i + j)th layer. Skip connections helps the network to understand global features as well.

Due to proven results of ResNets in image vision tasks, we will do a series of experiments using ResNet for our classification problem.

a. Experiment 7

Hypothesis: ResNet provides better predictions in our task.

ResNet Architecture: We will use the classical ResNet50 architecture. The paper can be found at https://arxiv.org/pdf/1512.03385.pdf

Total params: 24,278,100
Trainable params: 24,224,980
Non-trainable params: 53,120

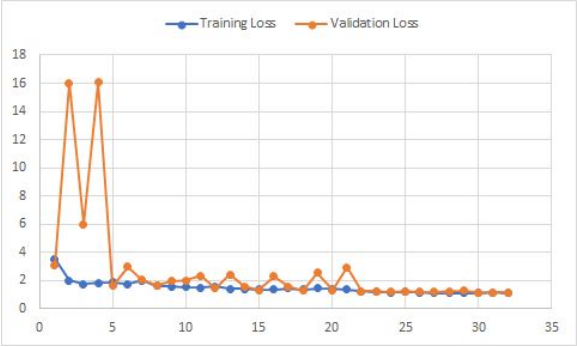
Hyperparameters:
Image Input size = 64x64
Steps per Epoch = 800
Batch size = 256
Epochs = 32
Initial Learning Rate = 0.002

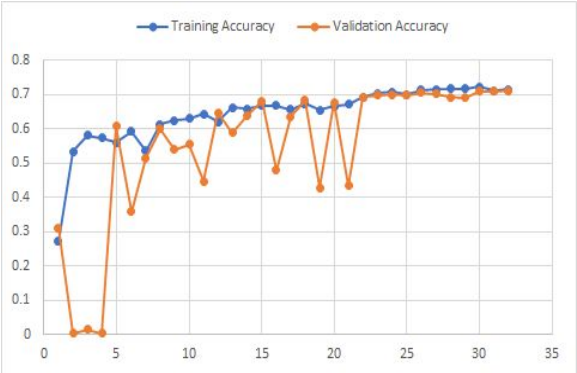
Subsequent strokes shade reduction = 13
Number of strokes considered = 10

Reasoning for architecture and hyperparameters selection
All the hyperparameters were tuned to accomodate to the computational limitations we faced since ResNet50 is very complex.

Our objective was to observe how ResNet performs on such a highly computationally constrained environment.

Performance
Map3 score on test set : 0.780
Kaggle Leaderboard Score : 0.809





Understanding the fit of data
The performance curves were generated by training the network on a smaller batch size of 256 and heavily downsampled images.

Limited by these constraints, the fit on the data seems to be very impressive.

Inferences / Conclusion
It is not possible to compare the performance of this experiment with the previous experiments since the control variables and the hyperparameters are different.

But in such a heavily computationally constrained environment, ResNet50 provides very impressive results. In the next experiment, we will do some hyper parameter tuning for ResNet, scale it identical to experiment 6 and observe the results.

b. Experiment 8 (Big Data)

Hypothesis: Training on larger dataset and providing sufficient computational resources can allow ResNet perform better than MobileNet in predictions.

Data: This is the second big data experiment that we will carry out. In this experiment we will use 17 million images for the experiment where we will use 98% for training and 2% for validation sets identical to experiment 6.

ResNet Architecture: Identical architecture as experiment 7

Hyperparameters:
Image Input size = 128x128
Steps per Epoch = 1000
Batch size = 256
Epochs = 100

Initial Learning Rate = 0.002
Subsequent strokes shade reduction = 13
Number of strokes considered = 10

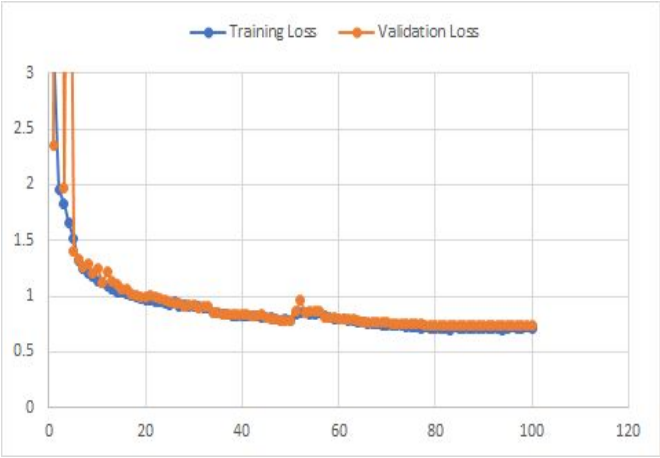
Reasoning for architecture and hyperparameters selection

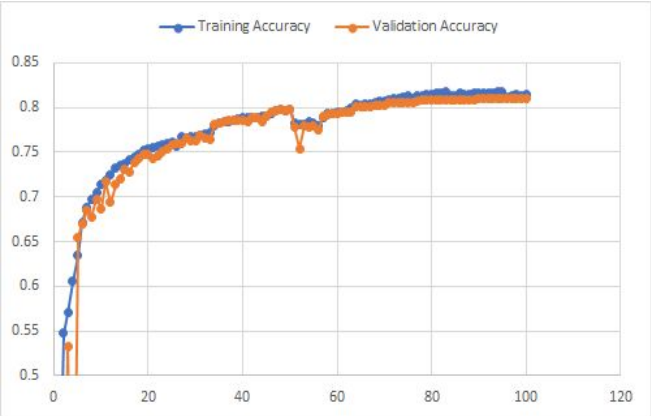
At this point of time, we decided to move to Google Cloud Infrastructure since we did not have access to reasonably powerful GPU clusters.

This allowed us to downsample the image to 128x128. Batch size was still kept at 256 due to cloud machine GPU limitations and therefore the number of epochs were doubled. All other hyperparameters are identical to experiment 6. (Big Data experiment 1)

Performance

Map3 score on test set : 0.878
Kaggle Leaderboard Score : 0.915





Understanding the fit of data

The experiment provides very good fit on the data.

Inferences / Conclusion

This experiment provides 1.3% improvement in predictions on our test data and 2.6% improvement on the kaggle test set compared to experiment 6. (Big Data experiment 1)

This a great improvement. Thus we have sufficient evidence to accept our hypothesis that ResNet50 performs better than MobileNets in our image classification problem.

Model and Hyper-parameter Selection

Based on the series of experiments, we have sufficient evidence to say that ResNet50 performs very well compared to MobileNet. We did not have sufficient resources to experiment with batch sizes for ResNet, but we do have sufficient insights on the noise distribution of the labels, learning rates, image downsampling ratios and color based stroke time-information encoding.

In the next section, we will carry out our final experiment to test our second image generator using the noise removed data.

a. Experiment 9 (Big Data)

Hypothesis 1: Including temporality for the first 10 strokes and gradually decaying stroke information after the 10th stroke will allow to learn the features better.

Hypothesis 2: Noise removed dataset using entropy boundaries of 0.03% (lower bound) and 98.5% (upper bound) will allow the network to learn better features.

Data: This is the third (last) big data experiment that we will carry out. We will use the same data used for experiments 6 and 8, but removal of noise scales down the dataset by 2.5%

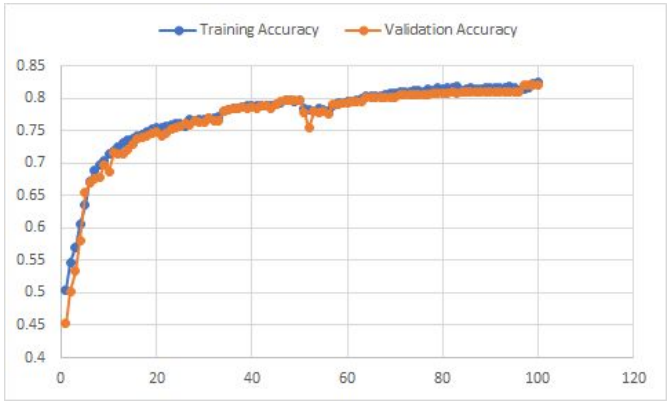
ResNet Architecture: Identical architecture as experiment 8

Hyperparameters:
Image Input size = 128x128
Steps per Epoch = 1000
Batch size = 256
Epochs = 100
Initial Learning Rate = 0.002
Subsequent strokes shade reduction = 13
Number of strokes for preserving temporality = first 10 strokes

**Do note that we are using a different version of an image generator in this experiment.

Reasoning for architecture and hyperparameters selection
Identical to experiment 8. Also the hyperparameters related to strokes were also kept the same so that the results can be interpreted easily.

Performance
Map3 score on test set : 0.891
Kaggle Leaderboard Score : 0.918



Understanding the fit of data
The experiment provides very good fit on the data.

Inferences / Conclusion
This experiment provides 1.5% improvement in predictions on our test data and 0.3% improvement on the kaggle test set compared to experiment 8. (Big Data experiment 2)

This a great improvement. Thus we have

sufficient evidence to accept our hypotheses.

Problems Faced

Processing time

Due to the the large dataset of 50 million points, testing,preprocessing and experimenting on a full dataset proved very time costly. So instead we conducted experiments on a subsample of the data and assume that the results we see on a smaller dataset would still hold when we scale up our data.

Noisey Labels

This dataset consist of extremely noisy data from false positives to false negative, that heavily affects the models ability to converge during training.

Training resources

The time taken to test each hypothesis in isolation took a very long time as we were computationally constrained to free resources such as Google colab. Due to the constraint of time we could not test all our hypothesis and instead we decided to prioritise and only test hypothesis that we thought would make the most difference.

Infrastructure

We initially started off training the kernels provided in kaggle. However due to their time limit of 6 hours we moved to Google Colab, which has a time limit of 12 hours. We conducted most of experiments on Google Colab and stored our data on an attached google drive.

For our final training we needed a more powerful GPU that would be capable of handling a higher batch size and image size and that would allow us to train for longer than 12 hours. As we were financially constrained we selected the Nvidia p100 GPU as it provided us a good trade of between performance and price. We used our own credits to train our model on the Google Cloud.

Conclusion

This project was a great learning experience with steep learning curves. This project also provided us opportunities to appreciate some prominent works done in the computer vision field.

Time constraint was a pressing problem and strategic compromises were considered in order to make effective progress.Source code and model weights can be found at https://drive.google.com/drive/folders/1KAky7fnDR-t1L7VV5vdWfZ4QjkRtf5_I

References

Benchmark Analysis of Representative Deep Neural Network Architectures
https://arxiv.org/pdf/1810.00736.pdf

SketchMate: Deep Hashing for Million-Scale Human Sketch Retrieval
http://openaccess.thecvf.com/content_cvpr_2018/CameraReady/2763.pdf

Local Shannon entropy measure with statistical tests for image randomness
https://www.sciencedirect.com/science/article/pii/S002002551200521X