Hand-Drawn Doodle Recognition

Mathew Sam **Ian Colbert**

Edward Zhang **Jared Leitner** **Apurba Bose**

Abstract

With recent advancements in Deep Learning, the field of image processing has seen vast improvements. From image captioning to super resolution, traditional Machine Learning tasks have seen significant advancements enabled by cutting edge Deep Learning algorithms, architectures, and acceleration hardware. Our work aims to leverage various models in recognizing and predicting words associated with hand-drawn doodle images. The images are pulled from the 'Quick Draw Dataset', a collection of millions of drawings across 300+ categories, contributed by players of Quick Draw [1]. Each drawing in the dataset is captured as a timestamped vector, tagged with metadata including what the player was asked to draw and in which country the player was located. The scope of our work utilizes subsets of these categories to compare the effectiveness of time-series modeling and image classification in hand-drawn doodle recognition.

1 Introduction

The field of image classification has exploded as data has become more readily available. From the MNIST dataset of 70,000 handwritten digits to the ImageNet dataset of over 14 million RGB images, computer vision and image processing have seen vast improvements in performance and generalizability.

As an open source project, Google AI released a dataset containing 50M drawings of 340 categories from their game "Quick, Draw!", in which participants were asked to hand draw images from certain categories [1]. This dataset is available in two forms: 1) The 'raw' dataset which contains the exact user input, i.e. the exact positions of the participants pen stroke sampled at a certain rate, and 2) the 'simplified' dataset which only contains the positions required to reproduce the image. These points correspond to the beginning and end of lines in the drawing, so that all the points in between can be reproduced by simply connecting the end points. This 'simplified' dataset is significantly smaller in size and still retains all relevant information.

The scope of our project uses the simplified dataset to compare the effectiveness of various deep neural network architectures and algorithms in classifying hand-drawn doodles. Each image is represented by a series of timestamped vectors, where each vector corresponds to a single stroke. Because we can represent the doodle images as both images and timestamp vectors, we can effectively compare models designed for both image classification and sequential modeling in hand-drawn doodle recognition. The challenges associated with this dataset stem from the fact that these are real hand drawn images from the game. As a result, a portion of the images may be incomplete and/or poorly drawn.

1.1 Motivation

In the scope of our work, we compare the effectiveness of an image processing approach and the approach of sequence modeling to correctly classify a doodle. We use CNNs (Convolutional Neural Networks) to learn effective filter banks that can extract discriminating features. With a deeper

network, it is possible to learn more complex features. However in the image space, the images are very simple and hence an overly complex model will overfit. By the same token, we compare this performance with a sequence model using LSTM (Long Short Term Memory) units. The sequence data is extremely complex being an extremely long sequence of sequences, thereby, necessitating a more complex model. The motivation behind the project is to decide the most effective way to tackle a problem that can be represented as sequences and as images.

To build fair comparisons, we evaluate the performance of different architectures with similar amounts of trainable parameters. We vary the strength of each network as well as the amount of classes we fit the model to. The rest of the paper is organized as follows: Section 2 dives into architecture selection and development. Section 3 describes the data in more detail. Section 4 presents our results and Section 5 concludes the paper.

2 Architectures and Algorithms

The fields of image classification and time-series analysis rarely overlap. Our project finds itself in the unique position to effectively compare the performance of both approaches on the same task, representing hand-drawn doodles as both 32x32 images as well as a series of time stamped vectors.

Due to their ability to model hierarchical representations of images using learned filter banks, convolutional neural networks [2] have consistently shown state-of-the-art performance on traditional machine learning tasks such as image recognition, object detection, and super resolution [3][4][5]. Designed to effectively mimic brain connectivity and visual processing methods, these locally connected networks possess desirable qualities that would prove effective in hand-drawn doodle recognition such as translation and scale invariance and robustness to noise [4].

Recurrent neural networks aim to model sequences by capturing the state of a given node and using it as input information for determining the next state [6] . However, this controlled feedback loop proved to introduce complexity to the exploding and vanishing gradient problem, which led to the introduction of the long short-term memory architecture [7][6].LSTM units provide a means for the model to propagate gradients through time using the memory cells present in LSTM units and thereby avoid issues pertaining to vanishing gradients. The non-linearities used within LSTM cells prevent exploding gradients.

2.1 Convolutional Neural Network

For our CNN models, we represent hand-drawn doodles as 32x32 images that can be used as inputs. We evaluate the performance of two architectures to study the complexity of the data and to see which approach works better. It was difficult to choose the right architecture for the problem since the images are sparse grayscale 32x32 images. To make the selection easier, we use a simple structure to start with and increase the complexity.

2.1.1 Vanilla CNN

The first is a vanilla CNN with three convolutional and max pooling layers followed by three fully connected layers. The convolutional layers use 3x3 windows while the maxpooling layers use 2x2 windows. The second is a more complex architecture with a convolutional layer followed by an inception module followed by another convolutional layer and then two fully connected layers. [8][9]3x3 windows are again used for the convolutional layers, and the fully connected layers reduce the number of the units to the corresponding number of classes.

2.1.2 Inception CNN

An inception module enables multiple convolutions of varying size to run in parallel, allowing for more features to be learned at different scales. This provides the model more flexibility to determine for itself what size convolutional windows are best for extracting features. The multiple feature maps are then concatenated and scaled to the correct size for the final convolutional layer. The inception module in our network uses 1x1, 3x1, 1x3, and 3x3 windows along with different padding techniques to test a variety of configurations. In addition, the inputs to each feature map are normalized using batch normalization in order to control the distribution of the inputs and reduce overfitting. Since this

model is more powerful in its ability to extract relevant features, we expect it to perform better at classification than the vanilla CNN.

2.2 Long-Short Term Memory

For our LSTM models, we represent hand-drawn doodles as sequential time-stamped vectors that can be used as inputs. Similar to our CNN analysis, we explore the use of two architectures. The first uses a standard encoding layer on top of stacked LSTMs followed by two fully-connected layers. Our second uses a series of 1D convolution filters on top of stacked LSTMs followed once again by two fully-connected layers [10].

For both models, the original data is first processed to stack all strokes together and then padded into the same length of 100. To preserve the information of different strokes, the stacked strokes also include a binary feature besides the original x-y coordinates to record the starting point of each stroke (1 for points at the start of each stroke, 0 otherwise).

2.2.1 Vanilla LSTM

For the vanilla LSTM, input sequences are directly fed into 2 stacked LSTM layers with 128 and 512 units respectively. The first LSTM layer is set with true return sequence, which means each input sequence will have correcsponding LSTM outputs. The second LSTM layer's return sequence is set to false, subsequently only one set of LSTM outputs. The LSTM layers are followed by 2 dense layers with 128 units and units with same number as classes from input respectively.

2.2.2 Conv LSTM

For the Convolution LSTM model, it has 3 extra Conv1D layers. The first two each has 48 and 64 channels of size 5x1 filters. The third Conv1D layer has 96 channels of 3x1 filters. The 2 LSTM and 2 dense layers have same structures as in vanilla LSTM.

3 Experimental Setting

3.1 Doodle Image Data

The data from the kaggle challenge consists of sequences of varying length describing the sketch, the country the sketch was drawn in, the label for the sketch and whether the sketch could be recognized by humans or not. The data was split into several .CSV files with each file corresponding to a specific label.There was a total of 340 classes. Each individual file was reasonably large, making it difficult to handle the data.

To handle this problem, data loader classes were constructed to load the data in different formats (as sequences or as images). The class would load blocks of data from different classes at a time to improve speed of preparing and loading the data. The loaded data would be shuffled and packaged into blocks and then converted to the desired format, either an image of size 32x32 or a padded sequence.

This problem is difficult because of the nature of the data and what passes as a sketch for different classes. There is a lot of variability in the data and it is not easy to generalize to a certain classes. For example, the contents of the class of crown images and mountain images show high intra-class variance and low inter-class variance.

3.2 Learning parameters

In this section, we describe the parameters used to train the CNN and LSTM models.

3.2.1 Convolutional Networks

Both the vanilla and inception CNN are trained using cross-entropy loss and mini-batch gradient descent, with a batch size of 150 samples over 10 epochs. A learning rate of 0.001 is used along with a momentum constant of 0.9. This momentum parameter introduces the idea of inertia into the gradient descent procedure, resulting in a less noisy gradient path.

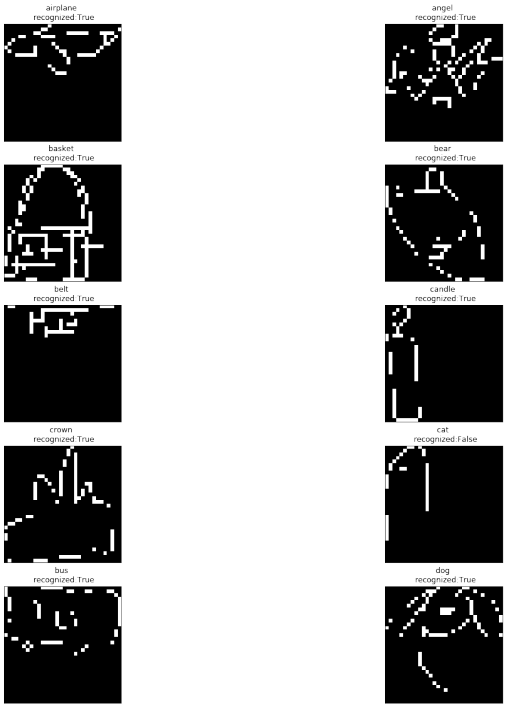

Figure 1: **Doodle Images.** Examples of Doodles Represented as images.

3.2.2 LSTM Networks

Both the models use Adam optimizer for mini-batch gradient descent with a batch size of 128 and categorical cross-entropy loss. After each LSTM layer the model applies dropout with p = 0.1 to prevent overfitting. It is trained for 640 epochs with 102400 samples from each class and tested with 4000 samples per class.

4 Results

In this section we compare the model performance of the vanilla and Inception CNN, as well as the performance of the vanilla and conv LSTM. The following figures show how the accuracy and loss of the different models improve during training and vary as the class size is increased. A final comparison between the best performing CNN and LSTM is made in order to deduce whether image classification or time series modeling is more successful for this problem.

4.1 Convolutional Neural Network

First, we will compare how the number of classes affects the performance of the vanilla and inception CNN. In both cases, as the number of classes increases, the performance of the model decreases. These results are expected since the network architecture remains the same for each class size. A larger number of classes results in a more difficult classification task, and without increasing the complexity of the model, it will perform worse. Each model is trained over 10 epochs and use a batch size of 150 when performing gradient descent. Although the number of epochs remain the same, the amount of data visited in each epoch increases with class size. The specific number of classes used are 8, 16, and 32, and Figures 2 and 3 display the results for each.

As observed by varying the number the number of classes, the performance of a vanilla CNN falls apart. This is because the difficulty of the problem drastically changes as the number of classes increases. As the number of classes increases, a more complex model is required to fit the data. The drop in accuracy is an indication of the degree of underfitting that occurs with the model and the data.

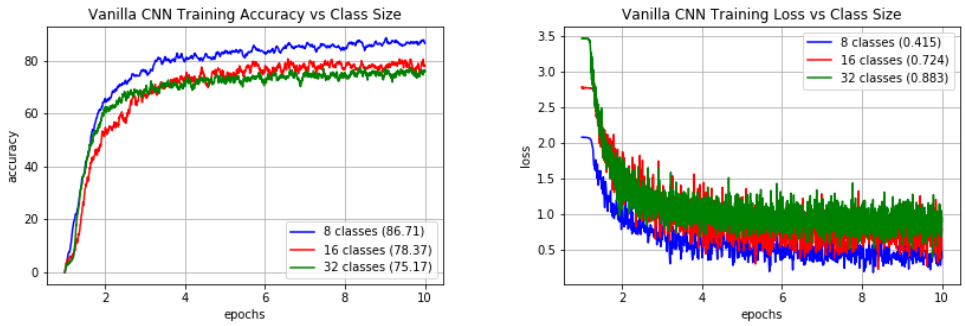

Figure 2: **Vanilla CNN Performance on Variable Amounts of Classes.**

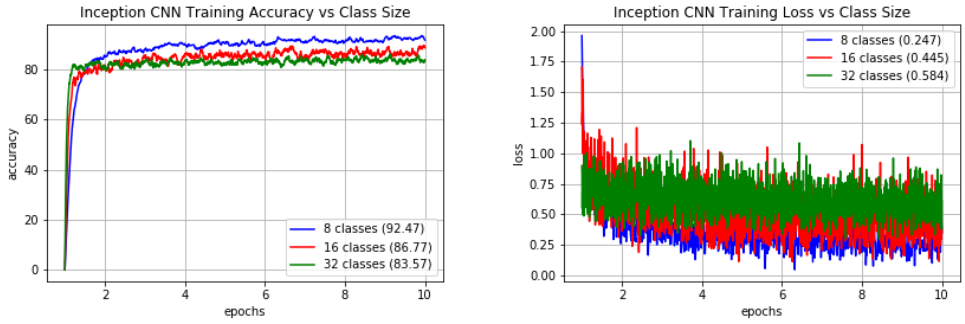

Figure 3: **Inception CNN Performance on Variable Amounts of Classes.**

The vanilla CNN achieves a training accuracy of 86.7%, 78.4%, and 75.2% for 8, 16, and 32 classes respectively.

A similar trend is observed for the inception CNN, as its performance is also worse for a larger number of classes. However, this architecture performs better than the simpler vanilla CNN which is evident in the following plots where they are directly compared for each number of classes.

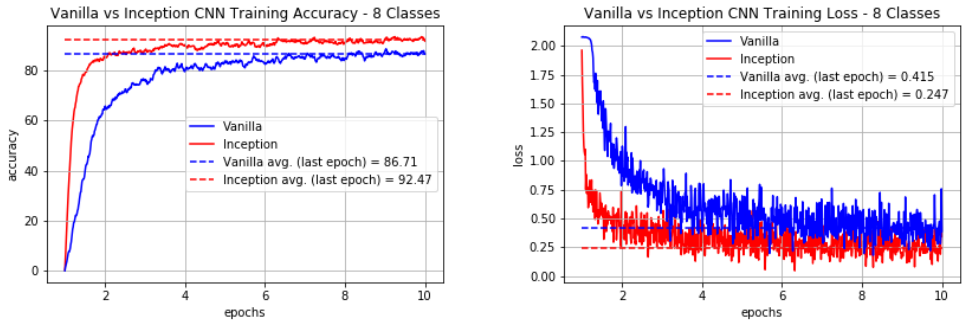

Figure 4: **Vanilla vs Inception Module CNN Using 8 Classes.**

The Inception CNN is far more complex and as such is capable of fitting the data much better and faster and also comes very close to fitting the data well. The inception model performs well with an increased amount of classes and therefore fits the model with 16 and 32 classes. The inception CNN achieves a training accuracy of 92.5%, 86.8%, and 83.6% for 8, 16, and 32 classes respectively.

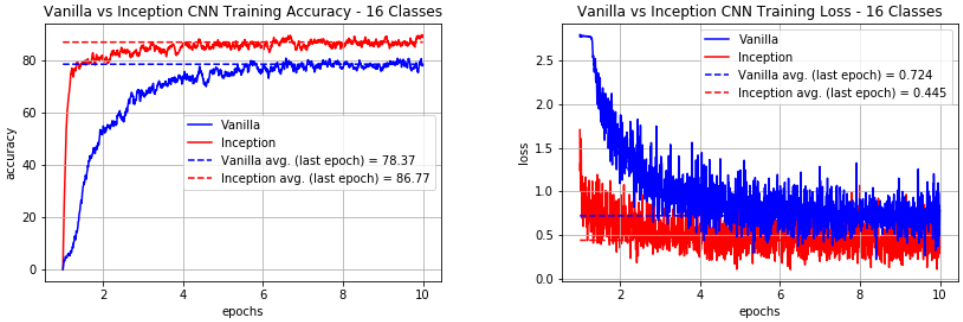

Figure 5: **Vanilla vs Inception Module CNN Using 16 Classes.**

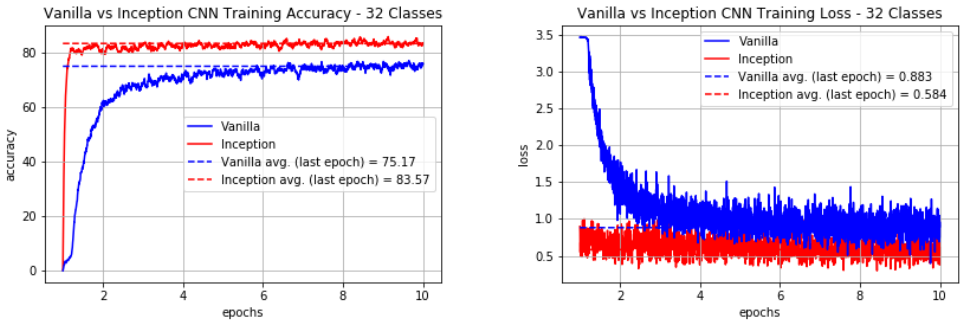

Figure 6: **Vanilla vs Inception Module CNN Using 32 Classes.**

4.2 Long Short-Term Memory

The graphs below show the training accuracy and loss of vanilla stacked LSTM model for 8, 16, and 32 classes. Since each epoch is trained with the same amount of data from each class, as class size increases, the model can learn general sequential features for all classes with more data. Thus it reaches saturation earlier. (The mini-batch size remains unchanged for different number of classes, 128. For larger classes, there are more steps for each epoch) Because the architecture remains unchanged for number of classes, we also observe an abnormal accuracy trend for low number class training, due to the model overfitting for small class-size case, as seen in the flat region for 8-class curves between epoch 20-100. Once the model is trained with enough data, it fits smaller class-size better than larger ones as expected. For our vanilla LSTM architecture, the model has a test accuracy of 93.9%, 91.9%, 91.8% for 8, 16, 32 classes respectively.

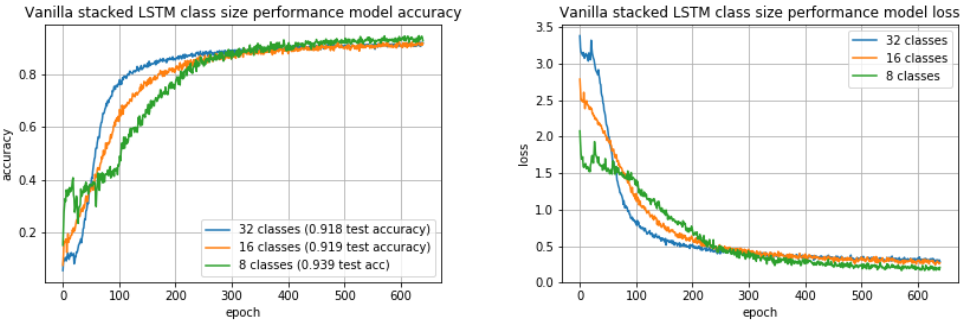

Figure 7: **Vanilla LSTM Performance on Variable Amounts of Classes.**

As for Conv LSTM model, all 3 cases with different class sizes reach saturation in training around the same epoch. Because the convolution layers learned feature mappings before the sequential layers, the LSTM layers can utilize information about strokes better. Notice that the effect of over-complex model still applies to the case with 8 classes. Once the model performance saturates, it shares similarities with For our vanilla LSTM architecture, the model has a test accuracy of 94.2%, 91.4%, 90.9% for 8, 16, 32 classes respectively.

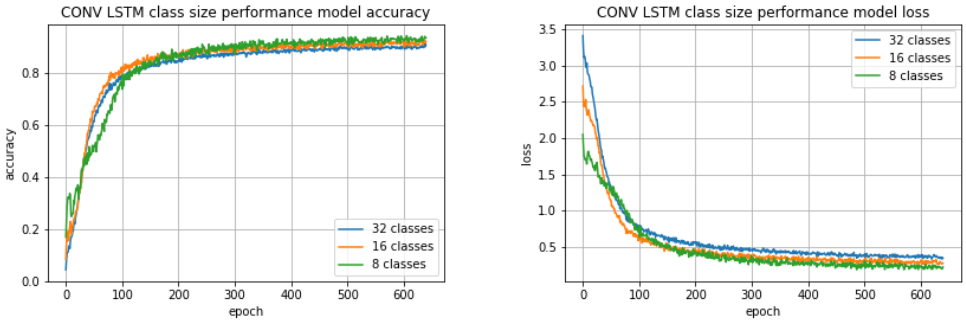

Figure 8: **Conv LSTM Performance on Variable Amounts of Classes.**

If we compare the performance of vanilla stacked LSTM model vs Conv LSTM model for each case of different class sizes, the effect of convolution layers preprocessing the sequences become more obvious. Not only the Conv LSTM model reaches saturation at more consistent epochs for different class sizes, but also earlier than the stacked vanilla model for all 3 cases. As class size increases, the gap between the two models decreases. The decrease can be interpreted as the sequential layers in vanilla stacked LSTM being able to learn more general stroke sequence features (features applied to all classes) as more data in each epoch is trained with.

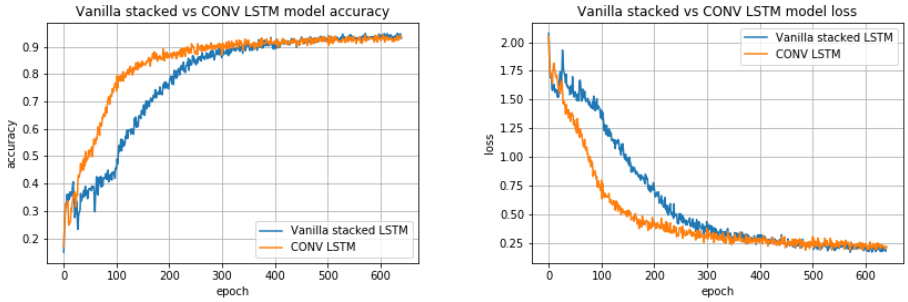

Figure 9: **8 Class Vanilla LSTM vs. Conv LSTM**

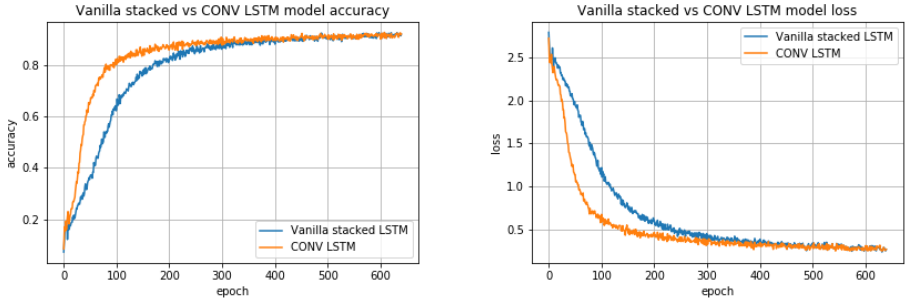

Figure 10: **16 Class Vanilla LSTM vs. Conv LSTM**

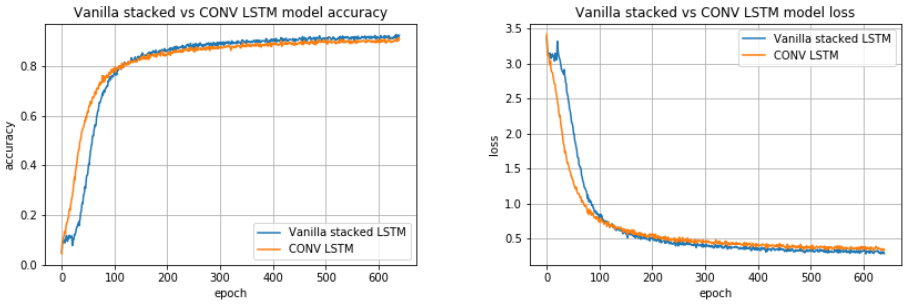

Figure 11: **32 Class Vanilla LSTM vs. Conv LSTM**

Considering the convolution layers used in our models add only around 7% (1.4 million vs 1.5 million) parameters, the Conv LSTM is the better choice. The following table summarizes the performance of each model.

	Vanilla CNN	**Inception CNN**	**Vanilla LSTM**	**Conv LSTM**
8 Classes	86.7%	92.5%	93.9%	94.2%
16 Classes	78.4%	86.8%	91.9%	91.4%
32 Classes	75.2%	83.6%	91.8%	90.9%

Figure 12: **Accuracy for each Model and Number of Classes**

4.3 Cases of Success and Failure

In this section we observe individual doodles that were correctly and incorrectly classified by both the inception CNN and conv LSTM.

4.3.1 Convolutional Architectures

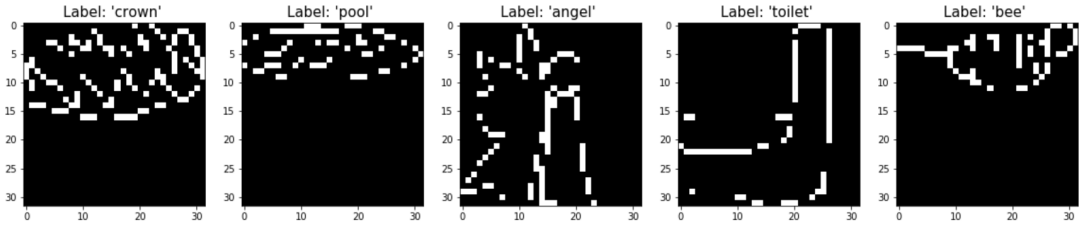

Figure 13: **Correct Classifications for Inception CNN Using 32 Classes.**

What is observed is that images that can be recognized by humans and have some intuitive link to what they represent are usually correctly classified. Images that are incorrectly classified are images that are very hard to differentiate even for humans and probably look like outliers in the predicted classes even though they belong to another class.

4.3.2 Sequential Architectures

The observations noted with the convolutional architectures can also be noted here. Most of the images in vector form are difficult to interpret and its harder to come up with conclusions here based

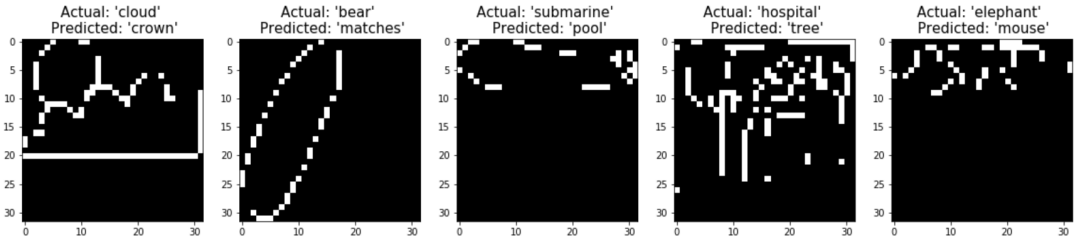

Figure 14: **Incorrect Classifications for Inception CNN Using 32 Classes.**

on correct and incorrect classifications. When the sequences are drawn out, they appear flipped due to the indexing used in the data set.

Figure 15: **Correct Classifications for LSTM Using 32 Classes.**

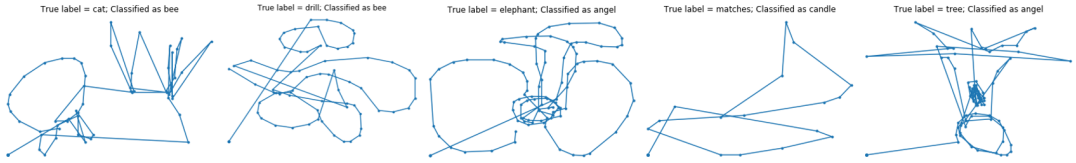

Figure 16: **Incorrect Classifications for LSTM Using 32 Classes.**

5 Discussion

5.1 Inferences

The data appears to be more complex when represented as sequences than as images, as expected. This inference can be drawn based on the number of epochs needed to train the model. The CNN models tend to saturate within 10 or so epochs while the sequence models take upwards of 600 epochs to train.

The variation in the complexity of the models can also be observed based on the loss accuracy plots. The complexity of the model needed exponentially increases with the number of classes available due to the relatively low inter-class variance and the high intra class variance. This is probably due to all the sketches drawn in the same scale. For example, a crown and a mountain range when sketched look similar because they both fill a 32x32 square. As a result complexity of the model needs to drastically increase with the number of classes.

In terms of speed of training, the CNN models seem to train much faster but the sequence models seems to do much better when it comes down to accuracy. This is probably attributed to the sparsity of the data. The padded sequence data tends to have much less sparsity and hence more information. The images, on the contrary, are mostly empty and exhibit a lot of sparsity. As such from a 1024 (32x32) vector, only about 100 dimensions have actual useful data and the positions of the 100 would

keep changing from class to class and instance to instance.The confusion expected while trying to classify between different classes would be more difficult and hence we see lower accuracy values.

We also see that the LSTM model which uses dimensionality reduction using 1D convolutions do better than vanilla LSTMs confirming that further dimensionality reduction could speed up the model and show better performance.

5.2 Conclusions

The conclusions drawn from the inferences above show that the images are a simpler representation of the data with much greater sparsity. The sequence representation has more information and is more dense. As a result, training for images are faster but training with sequences returns a higher accuracy as there is less unnecessary factors that could affect what the model is learning (instead of training with a 1024 dimensional data, we work with 100 dimensional data with sequences).

We can also conclude that the complexity of the model needs to increase with the number of classes as having more classes brings in greater difficulty differentiating between instances of the classes. However, dimensionality reduction is a viable option to simplify the data for classification. Successful dimensionality reduction will speed up the model while also increasing the accuracy as features that could possibly confuse the model are taken out of the problem.

5.3 Learning

From the above conclusions, we can state that while learning with sequences may be more beneficial from an accuracy perspective, for up-to 32 classes it is easier to work with images. However, if we use dimensionality reduction using an autoencoder or simply adding 1D convolutions, we can boost accuracy and performance.

5.4 Future work

Future work involves extending the training to all 340 classes.This brings in major issues since the current architecture may not scale well(as we have seen going from 8 to 16 to 32 classes). Training using sequence models, specially has its own drawbacks as training time for just 32 classes takes about 3 hours. One possible solution to better tackle this problem would be using Wavenet architecture to learn an image representation of the data and train a model on said image representation. This will provide a more dense representation of the data.

References

[1] Google AI: Doodle Recognition Challenge. https://www.kaggle.com/c/quickdraw-doodle-recognition/data, 2018-12-06.

[2] Alex Krizhevsky, I Sutskever, and G Hinton. Imagenet classification with deep convolutional neural. In *Neural Information Processing Systems*, pages 1–9, 2014.

[3] Charles Deledalle. Chapter iii - introduction to deep learning, 2018.

[4] Charles Deledalle. Chapter iv - image classification and cnns, 2018.

[5] Charles Deledalle. Chapter v - detection, segmentation, captioning, 2018.

[6] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Torchvision Github. https://github.com/pytorch/vision/blob/master/torchvision/models/inception.py, 2018-12-10.

[10] Jason Brownlee. https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/, 2018-10-10.

6 Appendix

- Github repo code: https://github.com/icolbert/MLIP285
- Github IO: https://icolbert.github.io/MLIP285/
- Hardware: Nvidia Geforce Gtx 1080i