Introduction

Many recent industry innovations have included neural networks. An important research area within neural networks has been image classification. One of the best techniques to classify images is Convolutional Neural Networks (CNN). We have sought to better understand industry best practices by exploring different image classification techniques with neural networks. Our exploration starts by manually performing image classification and noting the level of human accuracy. We then use a fully connected neural network to perform image classification before implementing a CNN. We implemented options for tuning and debugging the neural network while implementing each method during our process. We were able to develop an intuition for image classification best practices using neural networks after our research.

Many supervised machine learning methods often differ from those employed when constructing neural networks. These supervised machine learning methods, such as logistic regression or decision tree classification, require feature engineering to successfully classify an image. Models requiring feature engineering rely on considerable effort by subject matter experts to achieve reasonable accuracy. Neural networks allow for an approach which does not require feature engineering to achieve similar or better accuracy. Industry engineering efforts regarding feature engineering can be shifted to neural networks in some cases such as image classification. Moreover, the availability of cheap computing resources sped up a shift towards the use of neural networks.

When considering modeling approaches, we examine both fully connected and convolutional neural networks. A fully connected neural network is known to be inefficient at classifying large images. These inefficiencies in a fully connected neural network partly arise because every neuron in the $\ell-1$ layer must be connected to the ℓth layer. This requires a series of matrix multiplications and additions across each layer ℓ. Alternatively, a CNN allows for more efficient computations by: (1) efficient and automatic feature generation using local connectivity and parameter sharing of convolution operations, (2) dimensionality reduction using pooling layers. In this project, we attempt to understand these topics in depth from an empirical standpoint in relation to image classification using Google's ``Quick, Draw!'' dataset.

Methods

We generate hypotheses based on expectations related to model performance. We then conduct preprocessing steps on our data.

Overview of Statistical Modeling Techniques

We use human classification, fully connected neural networks, and a Convolutional Neural Network (CNN). We discuss the CNN in more detail than was covered in class so an overview is provided here.

Convolutional Neural Network (CNN)

TODO: explain how we used a CNN or anything else not covered in class

Hypotheses

We have generated the following hypotheses based on theoretical expectations:

Addressing model component of performance

```
\begin{align*}
  H1_a: &\text{Fully Connected Neural Network will struggle} \\
  &\text{ to get reasonable accuracy.} \\
  H2_a: &\text{CNN should perform better than fully} \\
  &\text{connected neural network.} \\
\end{align*}
```

Addressing data component of performance

```
\begin{align*}
  H3_a: &\text{Small amounts of data will lead to overfitting.} \\
  H4_a: &\text{Imbalanced class prediction can be improved}\\
  &\text{using data augmentation.} \\
  H5_a: &\text{Addition of a category to the trained model will} \\
  &\text{reduce the performance of the model.} \\
  H6_a: &\text{Adding more training data will improve the} \\
  &\text{performance of the model.} \\
\end{align*}
```

TODO: more context would be helpful here, maybe some citations

Data

Google originally collected the data from users by explicitly asking users to draw objects like forks, hotdogs, etc. The original dataset is 70 GB in size but Google also provides a simplified version of the dataset- consisting only of the final images. We used these '.npy' files to train all of our models. These numpy bitmap files consist of more than 100,000 rows each and 784 features, since they are basically 28×28 images. Samples of these images are shown in Figure \ref{fig:quickImages}. The subset of this dataset sampled varies from experiment to experiment. More details can be found in the experiments mentioned below.

```
\begin{figure}[h]
  \begin{center}
    \includegraphics[scale=0.5]{fig1}
  \end{center}
  \caption{Sample images from Quick, Draw! dataset}
  \label{fig:quickImages}
\end{figure}
```

Results

We report our results by examining the validity of each hypothesis.

Addressing model component of performance

In order to perform this experiment we built a fully connected neural network using tensorflow. Our dataset sample selected for this experiment had 10 categories: fish, fork, hotdog, flamingo, airplane, alarm clock, baseball bat, bicycle, dolphin, elephant. We had 100,000 examples in total, with 10,000 example for each of the aforementioned categories. Our train:test split was chosen to be 80:20, an industry standard.

Our initial objective was to build a model that overfit our given dataset, with the idea that once our model is complex enough to overfit the training data, we could regularize it in order to achieve a more optimal fit.

After experimenting with various architectures, we narrowed down to a 3 layer model as specified in Figure \ref{fig:fcnn}.

```
\begin{figure}[h]
  \begin{center}
    \includegraphics[scale=0.75]{fig2}
    \end{center}
  \caption{Fully Connected Neural Network Architecture}
  \label{fig:fcnn}
  \end{figure}
```

The training set accuracy observed was 93\%, while the test accuracy was at 73\%, which clearly indicated an overfit in the model, as designed.

The model took 1500 epochs to converge taking around ~45 minutes on a machine with 13 GB of RAM, Intel Xeon CPU 2.3 GHz (1 core, 2 threads), 1xTesla K80 GPU having 2496 CUDA cores with 12 GB GDDR5 VRAM. In order to manage variance, we attempted regularizing using dropouts. Unfortunately, that effort was futile. We then doubled the amount of data sampled to train the model, eventually helping us achieve training set accuracy of 86\% and test set accuracy of 82\%. Our learning rate was specified as 0.001.

While our initial hypothesis assumed that a regular fully connected network would struggle to achieve a reasonable accuracy, our experiment proved our hypothesis as invalid by achieving a commendable accuracy. Having said that, the model did take a longer to converge than expected.

As for the CNN, we started off with the same approach with attempting to overfit the model using convolution operations and a fully connected layer. Since our objective is to categorise 10 categories, softmax was chosen as the activation function for the final layer, while every other neuron had ReLU as the activation function owing to ReLU’s faster convergence.

```
\begin{figure}
  \begin{center}
    \includegraphics[scale=0.5]{fig3}
    \end{center}
  \caption{CNN Architecture}
  \label{fig:cnnArch}
\end{figure}
```

Loss Function: Categorical Cross Entropy [TO DO: Why?]
Optimizer: [TODO: Why?]
Number of Trainable Parameters: 320,890

The number of clearly being much greater than the number of observations used to train the model, we need to reduce the number of parameters vis-a-vis number of observations. From the available strategies to do so, we incorporated MaxPooling [TO DO: why?] post each convolution operation, in order to extract only the most significant features. As as a consequence, this reduced the volume of the output, giving the model a smaller memory footprint as well as a smaller number of trainable parameters.

```
\begin{figure}
  \begin{center}
    \includegraphics[scale=0.5]{fig4}
  \end{center}
  \caption{CNN Architecture with Reduced Parameters}
  \label{fig:cnnArchPool}
\end{figure}
```

The observable training accuracy is 0.90, whereas the validation accuracy is 0.89. The model converged after 10 epochs, which took ~ 3 minutes.

Some of the other unsuccessful strategies used to decrease the number of parameters and to regularize the network were strides and dropouts which sacrificed way too much in the way of accuracy and thus had to be dropped. Since most of our drawings had most of their information in the center of the image, padding, in theory wouldn’t be of much help. However, actually implementing padding improved our accuracy by 1\% at a cost of ~6000 parameters.

Conclusion

A slight (~5\%) improvement in the accuracy of a CNN when compared to a fully c\
onnected network, keeping the number of parameters constant, was observed. Howe\
ver, the CNN happened to be around 13 times faster in practice. This delta in a\
ccuracy and speed is likely to be amplified as we scale the model up with respe\
ct to model complexity or data.

Addressing data component of performance

Ok

Discussion

Talk about our final slide and tie the hypotheses together

Appendix

Dataset:

Google originally collected the data from users by explicitly asking them to draw objects like forks, hotdogs, etc. The original dataset is 70 GB in size but Google also provides a simplified version of the dataset- consisting only of the final images. We used these '.npy' files to train all of our models. These numpy bitmap files consist of more than 100,000 rows each and 784 features, since they are basically 28*28 images. The subset of this dataset sampled varies from experiment to experiment. More details can be found in the experiments mentioned below.

hotdog
fork
airplane
fish

Hypothesis:

We start off with hypothesis that we learnt from theoretical literature. We then perform several experiments to test the validity of these hypothesis and report out our findings. Some of our hypothesis is as follows:

1) Fully connected neural network will struggle to get reasonable accuracy. In contrast CNN's should perform better and faster than a Fully connected neural network.

2) Training a deep neural network with limited data will result in overfitting. Conversely, adding more data to a deep neural network is an effective way to manage variance.

3) Imbalance class prediction can be improved using data augmentation.

4) Increasing the number of categories will cause a significant drop in the performance of the neural network. Increasing the overall data points for the network to be trained on should help.

Methods:

Hypothesis 1: Fully connected neural network will struggle to get reasonable accuracy. In contrast CNN's should perform better and faster than a Fully connected neural network.

In order to perform this experiment we built a fully connected neural network using tensorflow. Our dataset sample selected for this experiment had 10 categories: fish, fork, hotdog, flamingo, airplane, alarm clock, baseball bat, bicycle, dolphin, elephant. We had 100,000 examples in total, with 10,000 example for each of the aforementioned categories. Our train:test split was chosen to be 80:20, an industry standard.

Our initial objective was to build a model that overfit our given dataset, with the idea that once our model is complex enough to overfit the training data, we could regularize it in order to achieve a more optimal fit.

After experimenting with various architectures, we narrowed down to a 3 layer model as specified below:
TODO: Add Network Diagram

$$Z_1 = \underset{25\times784}{W_1}\, X + \underset{25\times1}{b_1} \Rightarrow A_1 = \mathrm{ReLU}(Z_1)$$

$$Z_2 = \underset{12\times25}{W_2}\, A_1 + \underset{12\times1}{b_2} \Rightarrow A_2 = \mathrm{ReLU}(Z_2)$$

$$Z_3 = \underset{10\times12}{W_3}\, A_2 + \underset{10\times1}{b_3} \Rightarrow \text{Output Layer} \rightarrow \mathrm{SoftMax}(Z_3)$$

$$(25\times784 \underset{Z_1}{+} 25\times1) + (12\times25 \underset{Z_2}{+} 12\times1) + (10\times12 \underset{Z_3}{+} 10\times1) = 20,067 \text{ parameters}$$

The training set accuracy observed was 93%, while the test accuracy was at 73%, which clearly indicated an overfit in the model, as designed.

The model took 1500 epochs to converge taking around ~45 minutes on a machine with 13 GB of RAM, Intel Xeon CPU 2.3 GHz (1 core, 2 threads), 1xTesla K80 GPU having 2496 CUDA cores with 12 GB GDDR5 VRAM. In order to manage variance, we attempted regularizing using dropouts. Unfortunately, that effort was futile. We then doubled the amount of data sampled to train the model, eventually helping us achieve training set accuracy of 86% and test set accuracy of 82%. Our learning rate was specified as 0.001.

While our initial hypothesis assumed that a regular fully connected network would struggle to achieve a reasonable accuracy, our experiment proved our hypothesis as invalid by achieving a commendable accuracy. Having said that, the model did take a longer to converge than expected.

H2: CNN should perform better than fully connected neural network

As for the CNN, we started off with the same approach with attempting to overfit the model using convolution operations and a fully connected layer. Since our objective is to categorise 10 categories, softmax was chosen as the activation function for the final layer, while every other neuron had ReLU as the activation function owing to ReLU's faster convergence.

28 x 28 x 1 [Input Layer]	3 x 3 x 4 [ReLU] [Convolution]	2 x 2 x 8 [ReLU] [Convolution]	64 [Relu] [Dense]	10 [Softmax] [Output Layer]

Loss Function: Categorical Cross Entropy [TO DO: Why?]
Optimizer: [TODO: Why?]
Number of Trainable Parameters: 320,890

The number of clearly being much greater than the number of observations used to train the model, we need to reduce the number of parameters vis-a-vis number of observations. From the available strategies to do so, we incorporated MaxPooling [TO DO: why?] post each convolution operation, in order to extract only the most significant features. As as a consequence, this reduced the volume of the output, giving the model a smaller memory footprint as well as a smaller number of trainable parameters.

28 x 28 x 1 [Input Layer]	3 x 3 x 4 [ReLU] [Convolution] [MaxPool - 2x2]	2 x 2 x 8 [ReLU] [Convolution] [MaxPool - 2x2]	64 [Relu] [Dense]	10 [Softmax] [Output Layer]

The observable training accuracy is 0.90, whereas the validation accuracy is 0.89. The model converged after 10 epochs, which took ~ 3 minutes.

Some of the other unsuccessful strategies used to decrease the number of parameters and to regularize the network were strides and dropouts which sacrificed way too much in the way of accuracy and thus had to be dropped. Since most of our drawings had most of their information in the center of the image, padding, in theory wouldn't be of much help. However, actually implementing padding improved our accuracy by 1% at a cost of ~6000 parameters.

Conclusion

A slight (~5%) improvement in the accuracy of a CNN when compared to a fully connected network, keeping the number of parameters constant, was observed. However, the CNN happened to be around 13 times faster in practice. This delta in accuracy and speed is likely to be amplified as we scale the model up with respect to model complexity or data.

Hypothesis 2: Training a deep neural network with limited data will result in overfitting. Conversely, adding more data to a deep neural network is an effective way to manage variance.

With us reaching an optimal model for the dataset, we decided to study the effect of reduction in the sampled dataset and its relationship with overfitting.

Data(10 categories, epochs-10)	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
500	(0.39, 0.92)	(0.58, 0.82)	(1.93, 0.25)	(1.45, 0.63)
1000	(0.46, 0.91)	(0.62, 0.83)	(1.66, 0.28)	(1.22, 0.63)
3000	(0.66, 0.93)	(0.77, 0.87)	(1.09, 0.22)	(0.77, 0.87)
5000	(0.66, 0.93)	(0.79, 0.89)	(0.99, 0.22)	(0.66, 0.41)
7000	(0.69, 0.93)	(0.80, 0.89)	(0.99, 0.22)	(0.70, 0.38)
10000	(0.76, 0.91)	(0.83, 0.89)	(0.79, 0.20)	(0.56, 0.35)

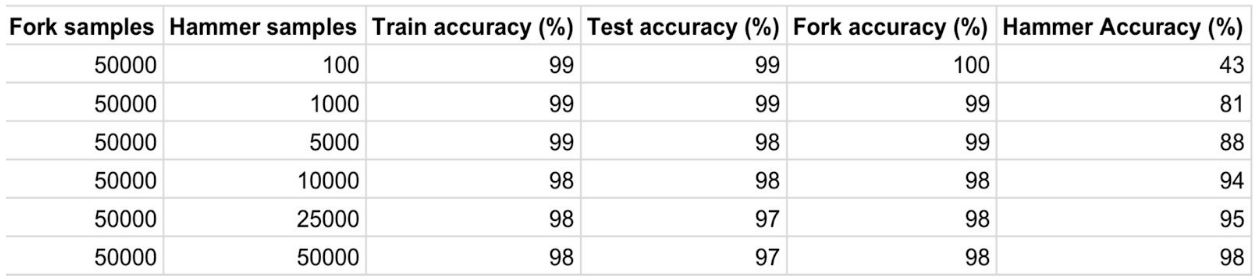
As shown in the table above, reducing the amount of data results in the model overfitting.

Hypothesis 3:

We tried to show the importance of having equal partitions of data to train a machine learning model. Imbalanced classes can make a model biased toward the majority class. We trained the neural network twice, on an imbalanced dataset, keeping the number of samples for the majority class (fork), fixed and decreasing the number of samples for the minority class (hammer).

The dataset was split into 80% and 20% for train and test respectively. This split had 0.2%, 2%, 10%, 20%, 50%, and 100% of majority class observations for the minority class.

We designed the test dataset with 20000 fresh samples from the main dataset to get a clearer picture of the model performance and we found that the accuracy on this fresh test dataset kept on increasing as the number of samples in the minority class increased.



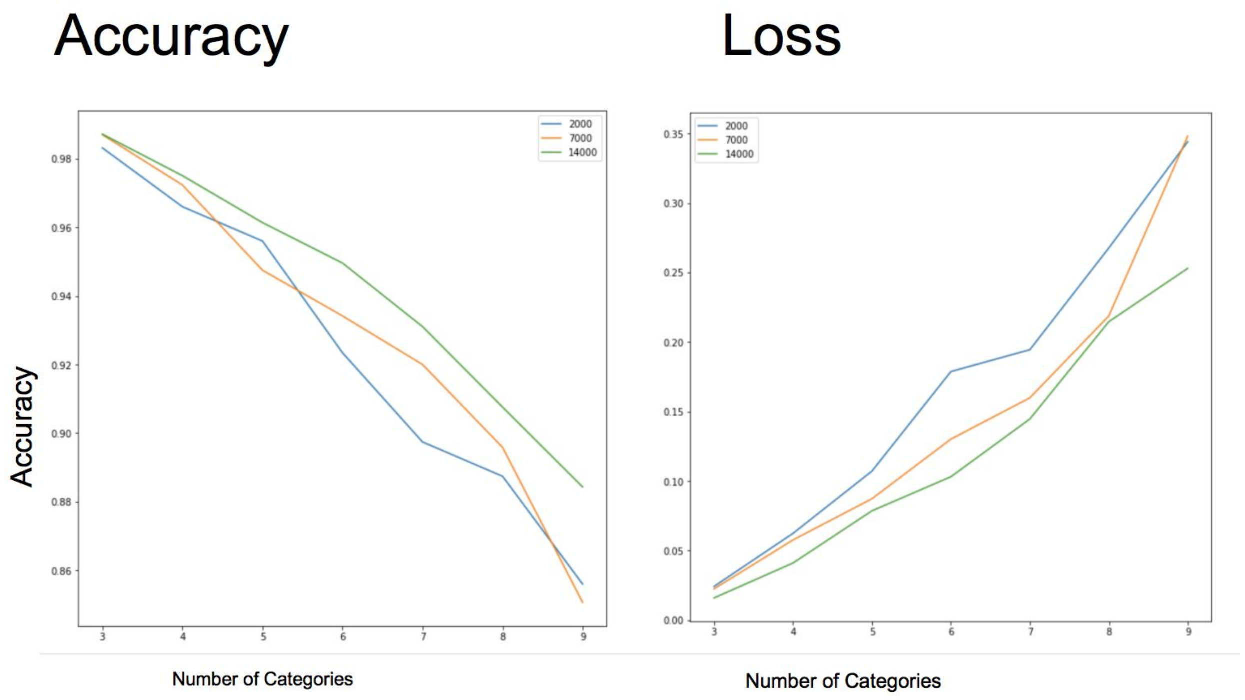
Fork samples	Hammer samples	Train accuracy (%)	Test accuracy (%)	Fork accuracy (%)	Hammer Accuracy (%)
50000	100	99	99	100	43
50000	1000	99	99	99	81
50000	5000	99	98	99	88
50000	10000	98	98	98	94
50000	25000	98	97	98	95
50000	50000	98	97	98	98

As shown in the figure above, we get unexpectedly high accuracy of 43% even in the worst case of 1:500 imbalance in the categories as neural networks tend to generalize well on the most important weights, while attempting to classify only the minority categories.

Hypothesis 4: Increasing the number of categories will cause a significant drop in the performance of the neural network. Increasing the overall data points for the network to be trained on should help.

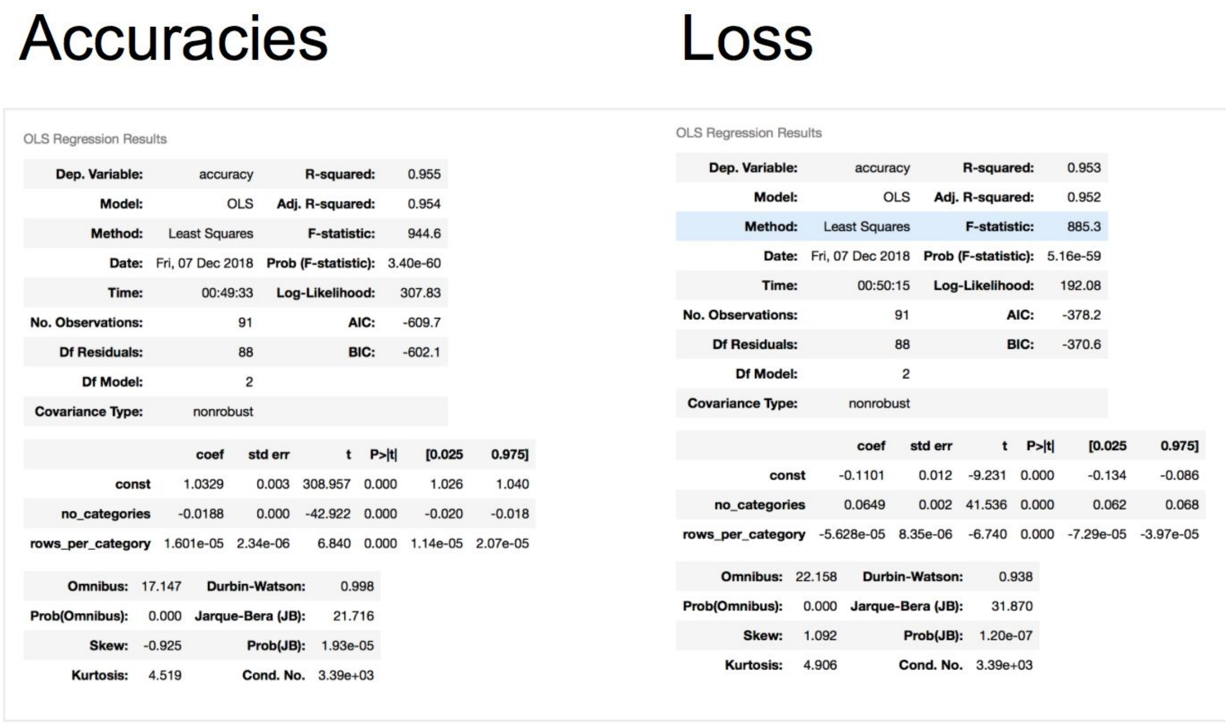
We try and understand the limits and capacity of the finalized model by varying the number of categories and the number of rows per category. We then measure the response accuracy and response loss of the model. The number of rows per category is altered in steps of 1000 from 7000 to 20000 and the number of categories is altered in steps of 1 from 3 to 10.

3 examples of the accuracy measurements with most rows (14000), least rows (2000) and intermediate rows (7000) and their performance per every additional category is plotted below.



As shown in the graph, it appears that the increase in the number of rows does not cause a significant improvement in the testing accuracy. On the contrary, as we increase the number of observations for each category, the decrease in the testing accuracy is relatively significant. While this is apparent from the graph, to put a number on it, we attempted to model the above dataset with OLS regression. The resulting parameters would talk to us about the effect of number of categories and the number of rows on the accuracy and loss.

Regression Results:



Accuracies

OLS Regression Results

Dep. Variable:	accuracy	**R-squared:**	0.955
Model:	OLS	**Adj. R-squared:**	0.954
Method:	Least Squares	**F-statistic:**	944.6
Date:	Fri, 07 Dec 2018	**Prob (F-statistic):**	3.40e-60
Time:	00:49:33	**Log-Likelihood:**	307.83
No. Observations:	91	**AIC:**	-609.7
Df Residuals:	88	**BIC:**	-602.1
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.0329	0.003	308.957	0.000	1.026	1.040
no_categories	-0.0188	0.000	-42.922	0.000	-0.020	-0.018
rows_per_category	1.601e-05	2.34e-06	6.840	0.000	1.14e-05	2.07e-05

Omnibus:	17.147	**Durbin-Watson:**	0.998
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	21.716
Skew:	-0.925	**Prob(JB):**	1.93e-05
Kurtosis:	4.519	**Cond. No.**	3.39e+03

Loss

OLS Regression Results

Dep. Variable:	accuracy	**R-squared:**	0.953
Model:	OLS	**Adj. R-squared:**	0.952
Method:	Least Squares	**F-statistic:**	885.3
Date:	Fri, 07 Dec 2018	**Prob (F-statistic):**	5.16e-59
Time:	00:50:15	**Log-Likelihood:**	192.08
No. Observations:	91	**AIC:**	-378.2
Df Residuals:	88	**BIC:**	-370.6
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.1101	0.012	-9.231	0.000	-0.134	-0.086
no_categories	0.0649	0.002	41.536	0.000	0.062	0.068
rows_per_category	-5.628e-05	8.35e-06	-6.740	0.000	-7.29e-05	-3.97e-05

Omnibus:	22.158	**Durbin-Watson:**	0.938
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	31.870
Skew:	1.092	**Prob(JB):**	1.20e-07
Kurtosis:	4.906	**Cond. No.**	3.39e+03

It can be interpreted from the results of the above regression that in for every additional category, we can expect the test accuracy go down by 1.88% and test loss to go up by 6.4% while for every additional datapoint, we can expect the accuracy to go up by .0016% and loss to go down by .0056%. Both results are statistically significant with reasonably tight confidence intervals and standard errors.

Discussion:

Our hypotheses explored how modeling choices and data distribution affect the accuracy performance. We found convolutional neural networks to be a good approach for classifying spatial data like images. Convolutional neural networks gave a better accuracy than humans which in turn was better than accuracy by fully connected neural networks.

In terms of modeling choices, Fully connected neural network(FC-NN) performed reasonably well for the Quick Draw dataset classification. We noted that Convolutional Neural network's(CNN) training and testing accuracy was 5 percent better than the FC-NN. Additionally, CNN was 13 times faster to train than FC-NN. These results highlighted the optimizations done by CNN with local connectivity and parameter sharing. The change in accuracy and speed is likely to be amplified as we scale the model up with respect to model complexity or data.

In terms of data distribution, we noted that for every additional category added to the finalized model architecture resulted in a 1.88 % decrease in test accuracy. Training the model on an imbalanced distribution of classes resulted in a poor prediction accuracy for the minority class. The prediction accuracy for the minority class increased as the imbalance in the training dataset decreased.

In future, we would like to make use of the time-series data for the drawings to predict the object while it is being drawn.

References:

1. Coursera Deep Learning Specialization
2. Dataset- Google Creative Labs

Appendix:

Fork samples	**Hammer samples**	**Train accuracy (%)**	**Test accuracy (%)**	**Fork accuracy (%)**	**Hammer Accuracy (%)**
100000	1000	99	99	99	63
100000	5000	99	99	99	85
100000	25000	98	98	99	90
100000	50000	98	98	99	94
100000	75000	98	97	97	97
100000	100000	98	97	98	98