Quick, Draw! Image Classification

Group 3

December 7, 2018

1 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 stanpoint in relation to image classification using Google's "Quick, Draw!" dataset.

2 Methods

Our methodology consists of reviewing statistical methodology, presenting theoretically informed hypotheses, and providing an overview of our data.

2.1 Overview of Statistical Modeling Techniques

We use human classification, fully connected neural networks, and a Convolutional Neural Network (CNN). We use more detail to describe techniques not covered during the course.

2.1.1 Understanding the Loss Function

Most modern neural networks are trained using maximum likelihood. This means that the cost function is the negative log-likelihood, also described as the cross-entropy between the training data and the model distribution [1]. In our use case, we are interested in correctly classifying 10 categories of drawings. Our output unit use a Softmax function in our output layer because they are most often used for classifiers representing the probability distribution over n different classes. Given features h, weights W, and bias b, a linear layer first predicts unnormalized log probabilities:

$$z = W^T h + b \tag{1}$$

where $z_i = \log \tilde{P}(y = i|x)$. The softmax function can then exponentiate and normalize z to obtain the desired $\hat{y}$. The softmax function is given by

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \tag{2}$$

As a cost function, we wish to maximize $\log P(y = i; z) = \log \text{softmax}(z)_i$. We can do this formally as

$$\log \text{softmax}(z)_i = z_i - \log \sum_j \exp(z_j). \tag{3}$$

which gives us the intuition for a single example. Overall, we want to learn parameters that drive the softmax to predict the fraction of counts of each outcome observed in the training set [1]:

$$\text{softmax}(z(x;\theta))_i \approx \frac{\sum_{j=1}^{m} 1_{y^{(j)}=i, x^{(j)}=x}}{\sum_{j=1}^{m} 1_{x^{(j)}=x}}. \tag{4}$$

This is how the categorical_crossetropy(y_true, y_pred) function is constructed in Keras [2] as well as the softmax_cross_entropy_with_logits_v2(labels, logits) in Tensorflow [3].

2.1.2 Understanding the Optimization Technique

This course discussed optimization algorithms such as Stochastic Gradient Descent ($m = 1$) and Batch Gradient Descent ($m = n$ obs.). Current best practices for Neural Networks recommend Mini-Batch Gradient Descent ($m = c$ batch size). In practice, we use the Adam optimization algorithm due to faster computation time. The Adam optimization algorithm is able to achieve a speedup due to its combination of two other techniques.

The Adam optimization algorithm takes an exponentially weighted average of the gradients, a technique called momentum, and then uses that gradient to update your weights. Secondly, we also want to slow down gradient descent in the horizontal direction with a technique called RMSprop to reduce oscillations from batch gradient descent. The Adam optimization method is combining momentum and RMSprop to perform gradient descent more quickly [4].

2.1.3 Convolutional Neural Network (CNN)

Convolutional networks, also known as convolutional neural networks(CNNs), are a specialized kind of neural network for processing data that has a known grid-like topology. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. See Figure 2 in the Appendix for a visual representation of a CNN.

CNN architectures are usually composed of the building blocks as described below:

1. **Convolution:** Convolution operation helps to extract features from the input image. The rst argument to the convolution is often referred to as the input , the second argument as the feature detector (kernel) and the output as the feature map. The convolution operation takes the dot product of feature map and the input to detect features. Since each convolution operation only focuses on a part of an image enabling parameter sharing which helps to reduce the number of free parameters compared to a fully connected neural network, allowing the network to be deeper with fewer parameters.
   - **Strides:** Stride is the number of pixels jumps between each convolution operation performed on the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. Increasing the stride parameter reduces the output volume of the convolution since the jumps are larger.
   - **Padding:**Padding helps to detect features present at the border by adding a dummy values around the edge. Since convolution operations collect most information from the center of the image, adding padding allows the border pixels to be weighed more during the convolution operation.

   See Figure 3 in the Appendix for an example of Stride and Padding.
2. **Pooling Layer:** Pooling layers section helps to reduce the number of parameters when the images are too large. Spatial pooling (subsampling or downsampling) reduces the dimensionality of each map, retaining important information. These can be of different types: (1) Max Pooling - takes the largest element (most important feature) from the rectified feature map (2) Average Pooling - averages all the pixel values (3) Sum Pooling. sum of all pixels values. See figure 4 for a visualization of a Pooling Layer.
3. **Fully Connected Layer:** The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network.

2.2 Hypotheses

We have generated the following hypotheses based on theoretical expectations expressed in the literature [1, 4]. Performance is effected by both the model and data [4] so our hypotheses pertain to both effects.

2.2.1 Addressing model component of performance

$H1_a$ Fully Connected Neural Network will struggle to get reasonable accuracy.

$H2_a$ CNN should perform better than fully connected neural network.

2.2.2 Addressing data component of performance

$H3_a$ Small amounts of data will lead to overfitting.

$H4_a$ Imbalanced class prediction can be improved using data augmentation.

$H5_a$ Addition of a category to the trained model will reduce the performance of the model.

$H6_a$ Adding more training data will improve the performance of the model.

2.3 Data

Google originally collected the data from users by explicitly asking users to draw objects like forks, hotdogs, etc [5]. The original dataset is 70 GB in size but Google also provides a simplified version of the dataset- consisting only of the final images [6]. 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 1. The subset of this dataset sampled varies from experiment to experiment. More details can be found in the experiments mentioned below.

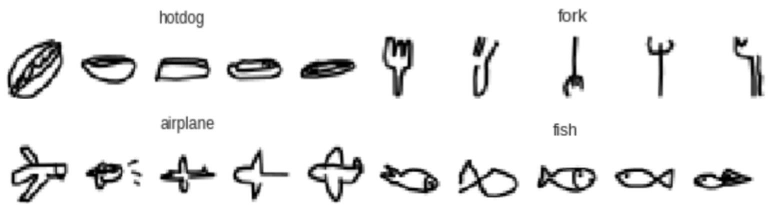


Figure 1: Sample images from Quick, Draw! dataset

Our analysis required us to use a subset of this data. 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.

2.4 Modeling Strategy

We provide our modeling strategy as it relates to the exploration of our stated hypotheses.

2.4.1 Baseline image classification using humans

Two group members completed a Human Intelligence Task by manually classifying images into one of our ten defined categories. This approach helped us get a benchmark which we could use to define “reasonable accuracy”.

2.4.2 $H1_a$: Fully connected neural network will struggle to get reasonable accuracy

In order to perform this experiment, we built a fully connected neural network using Tensorflow. 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. The fully connected neural network had 20,067 parameters.

After experimenting with various architectures, we narrowed down to a 3 layer model as specified in Figure 5 in the Appendix. The model was run 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.

2.4.3 $H2_a$: 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 with an objective 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. See Figure 6 for a representation of this first CNN model.

The number of parameters is much greater than the number of observations used to train the model resulting in overfitting. In order to reduce the number of parameters, we incorporated Pooling (Max Pooling) post each convolution operation extracting only the most significant features resulting in reduction of the volume of the convolutional output, giving the model a smaller memory footprint as well as a smaller number of trainable parameters. Figure 7 has a representation of this second CNN model. We employ strategies to reduce overfitting related to dropouts and stride in addition to adding more training data.

2.4.4 $H3_a$: Small amounts of data will lead to overfitting

Once we have found an optimal model for the dataset, we use this model to study the effect of reduction in the sampled data. The optimal model helps inform the relative performance of models trained with lower amounts of data.

2.4.5 $H4_a$: Imbalanced class prediction can be improved using data augmentation

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.

2.4.6 $H5_a$ & $H6_a$: Addition of a category to the trained model will reduce the performance of the model & Adding more training data will improve the performance of the model

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 categories. We attempt to formalize the patterns we see from visualizing the data with an OLS regression.

3 Results

We report our results by examining the validity of each hypothesis related to the model and data components of performance.

3.1 Baseline image classification using humans

Our baseline performance when manually classifying images into one of 10 categories is 87% accuracy on average. We consider this accuracy to be “reasonable accuracy”. The accuracy serves as a baseline for informing our understanding of results for our models.

3.2 Addressing model component of performance

These results pertain to hypotheses effected by modeling choices.

3.2.1 $H1_a$ Fully Connected Neural Network will struggle to get reasonable accuracy

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.

3.2.2 $H2_a$ CNN should perform better than fully connected neural network

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.

3.3 Addressing data component of performance

These results pertain to hypotheses effected by the underlying distribution of data.

3.3.1 $H3_a$ Small amounts of data will lead to overfitting

As found in the experiments, after decreasing the amount of data resulted in overfitting thereby confirming our hypothesis. Figure 8 contains a table with results for various number n of observations. We increased the number of categories the Neural Network had to model, holding the number of examples per categories constant. Observations were increased in steps of 1,000 from $n = 7000$ to $n = 20,000$ examples. The number of categories k ranged from $k = 3$ to $k = 10$ in steps of 1.

3.3.2 $H4_a$ Imbalanced class prediction can be improved using data augmentation

Results can be seen in the Appendix with Figure 9. 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 7,000 to 20,000 and the number of categories is altered in steps of 1 from 3 to 10.

3 examples of the accuracy measurements with most rows (14,000), least rows (2,000) and intermediate rows (7,000) and their performance per every additional category is plotted below.

3.3.3 $H5_a$ & $H6_a$ Addition of a category to the trained model will reduce the performance of the model/Adding more training data will improve the performance of the model

Results for various number of classes k are visualized in the Appendix under Figure 10. Results for various amounts of training data are under Figure 11 in the Appendix. It can be interpreted from the regression that 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.

4 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 the 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] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] François Chollet et al. Keras. https://keras.io, 2015.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Andrew Ng. Deep learning — coursera. https://www.coursera.org/specializations/deep-learning, 2018. (Accessed on 12/07/2018).

[5] Google. Quick, draw! doodle recognition challenge — kaggle. https://www.kaggle.com/c/quickdraw-doodle-recognition, 2018. (Accessed on 12/07/2018).

[6] Google. googlecreativelab/quickdraw-dataset: Documentation on how to access and use the quick, draw! dataset. https://github.com/googlecreativelab/quickdraw-dataset, 2016. (Accessed on 12/07/2018).

5 Appendix

Our project code is available in a GitHub repo: “https://github.com/tbonza/ds5220”.

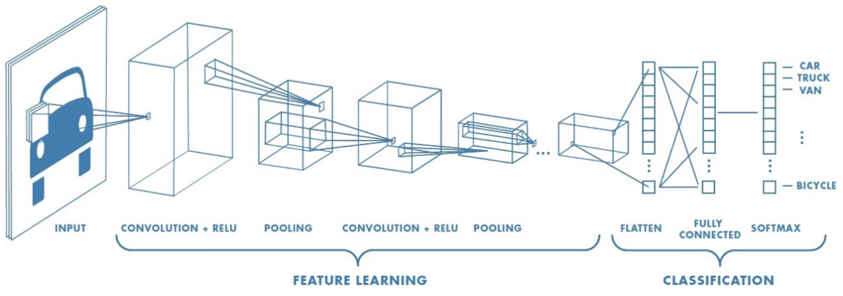


Figure 2: CNN Architecture Example

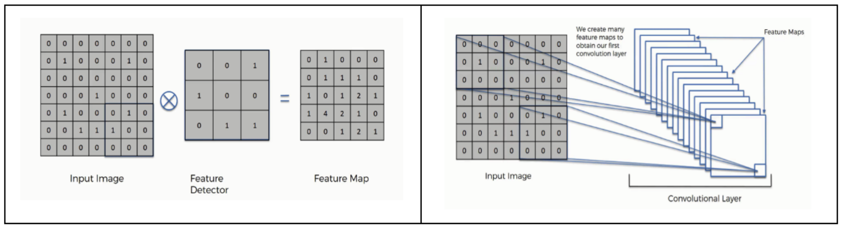


Figure 3: Convolution operation with stride=1, padding=None

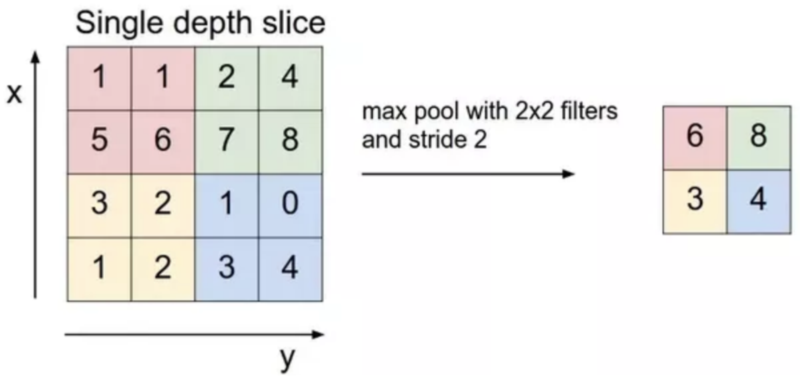


Figure 4: Example of Max Pooling

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

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

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

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

Figure 5: Fully Connected Neural Network Architecture

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]

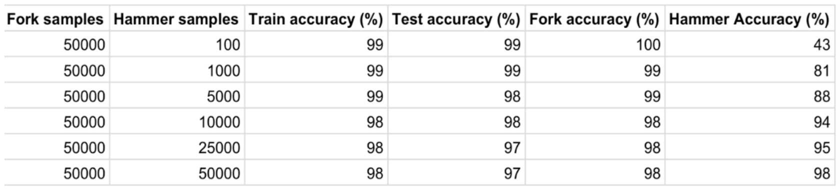
Figure 6: (CNN Model 1) Loss Function: Categorical Cross Entropy, Optimizer: Adam, Trainable Parameters: 320,890

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]

Figure 7: (CNN Model 2) Loss Function: Categorical Cross Entropy, Optimizer: Adam, Trainable Parameters: 19,322

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)

Figure 8: $H3_a$ Results for various size training data at (epoch 1, epoch 10)



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

Figure 9: $H4_a$ Results for various class imbalances

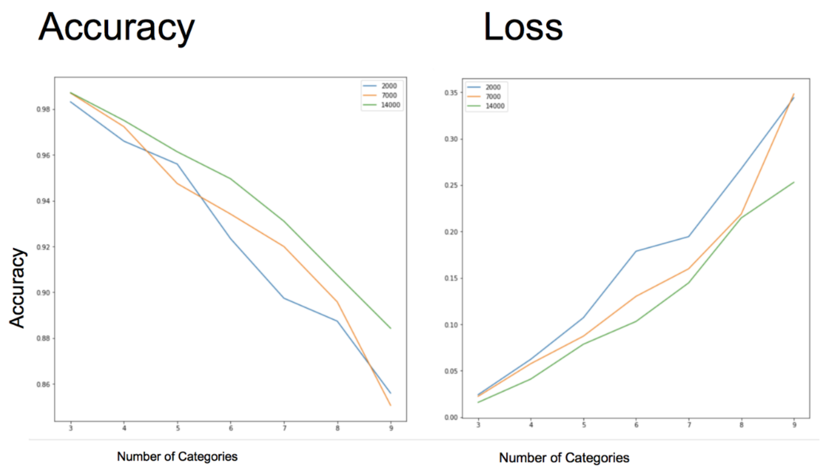
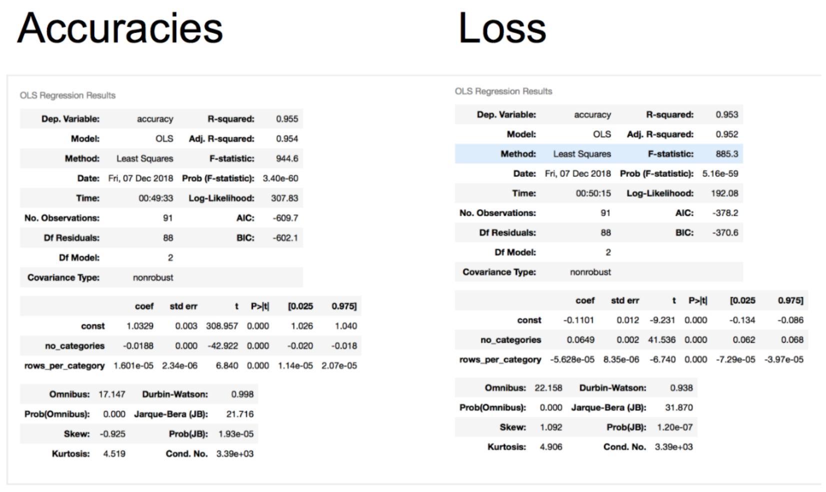


Figure 10: $H5_a$ Results for various number of classes k



```
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
```

```
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
```

Figure 11: $H6_a$ Results for various amounts of training data n