|  |  |  |  |
| --- | --- | --- | --- |
| D:\NEU\ADS\final_proj\images\english.png | Character Recognition using  Deep Learning   |  | | --- | |  | |  |   Prajakta Pawar  Rutuja Magar  Dhanshri More  INFO7390  - Srikant Krishnamurthy  19th August 2016 |

Topics Covered

|  |  |  |
| --- | --- | --- |
| 1 | Introduction  Libraries | http://blog.h2o.ai/wp-content/uploads/2015/08/h2o-home-e1439746470600.png |
|  |  |  |
| 2 | Convolution Net | https://tableau.lcsexams.com/images/TableauLogo.jpg |
|  |  |  |
| 3 | Tableau Analysis | https://pbs.twimg.com/profile_images/593514854893232132/IepDFTuO.jpg |
|  |  |  |
| 4 | Character Recognition   * Azure Pipeline * Confusion Matrix | https://www.enclout.com/assets/excel_2013-3e7309ea2dbd8944be164009d840feae.png |
|  |  |  |
| 5 | Handwritten Digit Recognition   * Azure Pipeline * Confusion Matrix * Excel Analysis – Scored Labels | http://4.bp.blogspot.com/-k2inbcOpdDs/Vb_OP5HZyyI/AAAAAAAABfM/Y1G_l37gjrU/s80-c/shiny_logo.png |

# Introduction:-

Recognizing arbitrary characters in unconstrained natural photographs is a hard problem. In this project, we address an equally hard sub-problem in this domain - recognizing arbitrary single characters from Street View images. In this project, we propose a unified approach that integrates both localization and segmentation via the use of convolutional neural networks that operates directly on the image pixels. We find that the performance of neural network solution works much better than the traditional approach of classification based on image features, while the performance of this approach increases little with the depth of the convolutional network.

For example, when a user enters a query into Google Maps, they are presented with a list of street level view for the address or target they have searched for. However, problems occur when a user searches for the street view image for a specific business or landmark, they prefer to see their object of interest centered in the Google Street View (GSV) viewport, instead of a list of un-oriented, un-organized street view images around the target or within an arbitrary distance from the target. In order to solve this problems, text recognition is the ideal solution as words recognized from the Google Maps search query can be identified in the nearby images so that the view can be centered upon an instance of text from the search query.

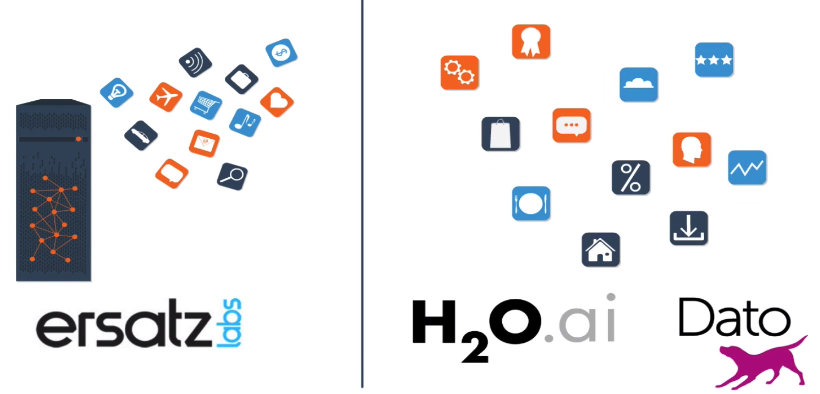


Figure 1: These are examples of individual characters that appear in the text of street view photography. We found that there is high visual similarity between samples of different classes caused mainly by the lack of visual context. For example, character ‘O’ and ‘0’, ‘1’, ‘I’ and ‘l’ are extremely similar in the real world street view images.

If there's one thing that gets everyone stoked on AI, it's Deep Neural Networks (DNN). From Google's pop-computational-art experiment, DeepDream, to the more applied pursuits of face recognition, object classification and optical character recognition Neural Nets are showing themselves to be a huge value-add for all sorts of problems that rely on machine learning.

### Libraries available in Deep Learning:-

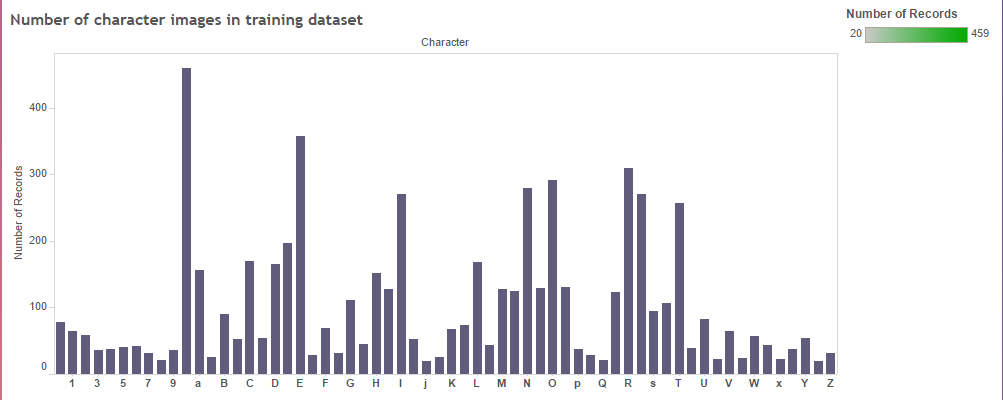




### Library used by us:-

* H20

### Tableau Analysis:-





# Character Recognition:-

# Business Case

We tried to approaches a different, but no less critical problem in the field: how to deal with noisy, real-world image data. In emerging fields like autonomous vehicles and computer-assisted sight, there is a great need for not only being able to "read" and convert optical input into strings, but also to also account for the immense variety in typefaces encountered on an average street.

The Chars74k dataset contains images of characters with varying backgrounds, orientations, image qualities and typefaces - which turns OCR into a much more interesting problem!

In this work, we intend to correctly classify single characters from street view images into 62 categories – ‘0~9’, ‘a~z’, ‘A~Z’.

**Exploratory Data Analysis:**

We frame this task as a classification problem where the eventual goal is to map the single characters to the correct type of the class. First, we label 62 categories of characters – ‘0~9’, ‘a~z’, ‘A~Z’ as numerical number 0~61 in the same order. For example, label 0 is mapped to character ‘0’, label 10 is mapped to character ‘a’, and label 61 is mapped to character ‘Z’. Then, the problem is, given an input image containing a single character, to find a model that could predict the probability/score of each class and the class with highest probability/score is the correct class for that character.

**Data Processing**

We used dataset from the public available Chars74K dataset, which includes 12483 images, covering 64 classes of letters (0-9, A-Z, a-z).

The 6283 training images are rescaled into size 20X20 matrix and have an associated label.

**Data Split**

We split the 12483 images into three subsets for training and testing

|  |  |
| --- | --- |
| Training | Testing |
| 6283 | 6200 |

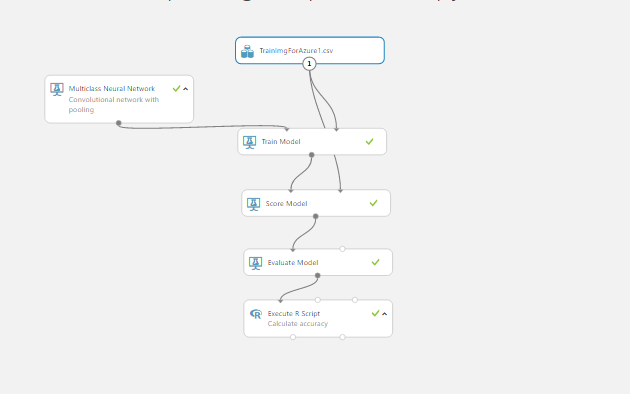
**Expected** **Results**

We built a classification model that predicts an unknown image (containing a single character) with probability/score values for each numerical label. The model which gives the best output is Convolutional Neural Networks (CNN).

**Evaluation**:

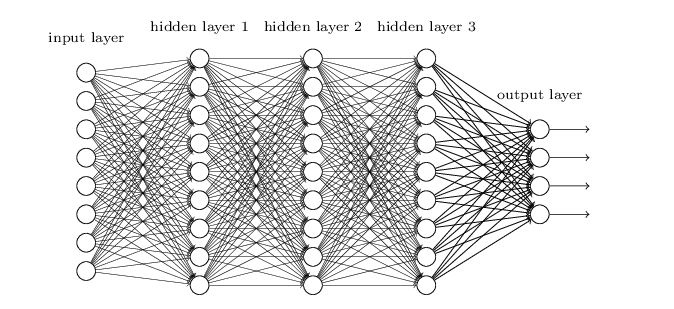
We will use 6200 withheld images in the test set to evaluate our method. The 6200 test images were pre-processed in the same way as those 6283 training images rescaled into size 20X20 and a class label. We won’t touch this test data unless the classifiers were developed. We will apply the trained classifiers on those 6283 withheld images and compare the predicted class label with the true class label. Finally, we mapped the predicted class label to the class name so that our pipeline could serve an application to recognize real-world characters.

# Technical Approach – Azure Model



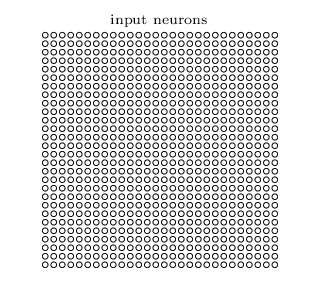
# Convolution Neural Network:-

We did this using convolution networks in which adjacent network layers are fully connected to one another. That is, every neuron in the network is connected to every neuron in adjacent layers.



**Local receptive fields:**

In the fully-connected layers shown earlier, the inputs were depicted as a vertical line of neurons. In a convolutional net, it'll help to think instead of the inputs as a 28×2828×28 square of neurons, whose values correspond to the 28×2828×28pixel intensities we're using as inputs:



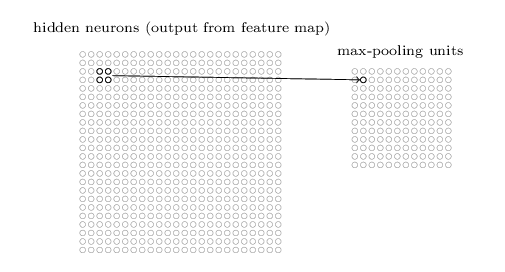
As per usual, we'll connect the input pixels to a layer of hidden neurons. But we won't connect every input pixel to every hidden neuron. Instead, we only make connections in small, localized regions of the input image.

To be more precise, each neuron in the first hidden layer will be connected to a small region of the input neurons, say, for example, a 5×55×5 region, corresponding to 2525 input pixels.

**Pooling layers:**

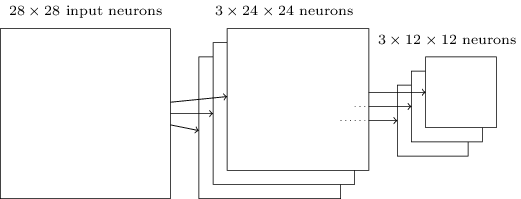
In addition to the convolutional layers just described, convolutional neural networks also contain pooling layers. Pooling layers are usually used immediately after convolutional layers. What the pooling layers do is simplify the information in the output from the convolutional layer.

In detail, a pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map. For instance, each unit in the pooling layer may summarize a region of (say) 2×22×2 neurons in the previous layer. As a concrete example, one common procedure for pooling is known as max-pooling. In max-pooling, a pooling unit simply outputs the maximum activation in the 2×22×2 input region, as illustrated in the following diagram



Note that since we have 24×2424×24 neurons output from the convolutional layer, after pooling we have 12×1212×12 neurons.

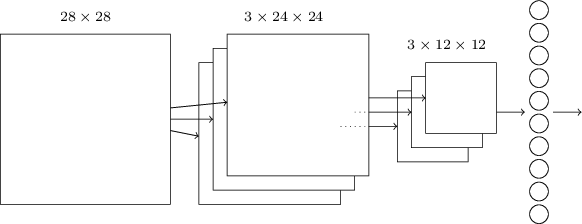
As mentioned above, the convolutional layer usually involves more than a single feature map. We apply max-pooling to each feature map separately. So if there were three feature maps, the combined convolutional and max-pooling layers would look like:



We can think of max-pooling as a way for the network to ask whether a given feature is found anywhere in a region of the image. It then throws away the exact positional information. The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features. A big benefit is that there are many fewer pooled features, and so this helps reduce the number of parameters needed in later layers.

**Putting it all together:**

We can now put all these ideas together to form a complete convolutional neural network. It has the addition of a layer of 10 output neurons, corresponding to the 10 possible values for MNIST digits ('0', '1', '2', *etc*):

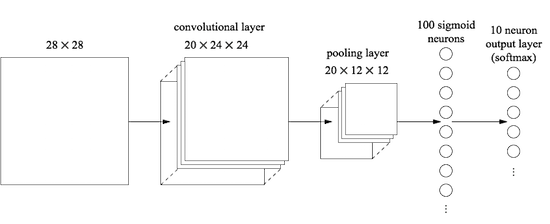


The network begins with 28×28 input neurons, which are used to encode the pixel intensities for the MNIST image. This is then followed by a convolutional layer using a 5×5 local receptive field and 5 feature maps. The result is a layer of 5×24×24 hidden feature neurons. The next step is a max-pooling layer, applied to 2×2 regions, across each of the 5 feature maps. The result is a layer of 5×12×12hidden feature neurons.

The final layer of connections in the network is a fully-connected layer. That is, this layer connects *every* neuron from the max-pooled layer to every one of the 10 output neurons. This fully-connected architecture is the same as we used in earlier chapters.

**Convolutional Network Parameters:-**

1. Number of learning iterations : 1000
2. Number of hidden layers : 500



**Sample Images:-**

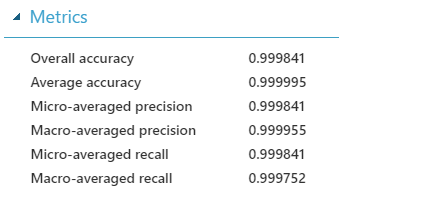
1)



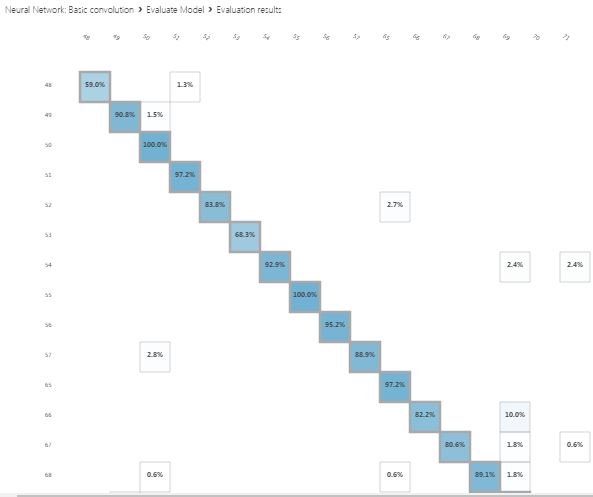
2)



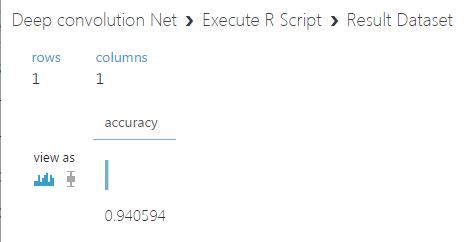
# Evaluation Results – Azure Model



**Confusion Matrix**



**Overall Accuracy:-**



# Handwritten Digit Recognition:-

Handwritten digit recognition plays a significant role in many user authentication applications in the modern world. As the handwritten digits are not of the same size, thickness, style, and orientation, therefore, these challenges are to be faced to resolve this problem.

# Business Case

We developed an artificial neural network designed to recognize handwritten digits. We used the MNIST database which has a training set of 60,000 examples and a test set of 10,000 examples of handwritten digits. The model uses Multiclass Decision Forest algorithm for better accuracy.

The dataset contains images of digits with varying height, width, image qualities and orientation - which turns this into a much more interesting problem!

In this work, we intend to correctly classify single characters from street view images into 10 categories – ‘0~9’.



**Problem Description:**

We frame this task as a classification problem where the eventual goal is to map the single characters to the correct type of the class. First, we label 10 categories of characters – ‘0~9’ as numerical number 0~9 in the same order. For example, label 0 is mapped to character ‘0’, label 9 is mapped to character ‘9’. Then, the problem is, given an input image containing a single digit, to find a model that could predict the probability/score of each class and the class with highest probability/score is the correct class for that digit.

**Data Processing**

We used dataset from the public available MNIST which includes 70000 images, covering 10 classes of digits (0-9).

The 60000 training images are rescaled into size 28X28 matrix and have an associated label.

**Data Split**

We split the 70000 images into three subsets for training and testing

|  |  |
| --- | --- |
| Training | Testing |
| 60000 | 10000 |

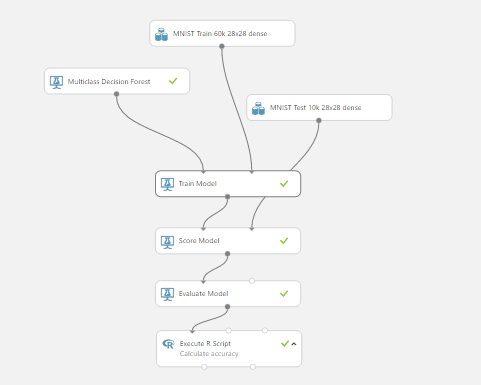
**Expected** **Results**

We built a classification model that predicts an unknown image (containing a single digit) with probability/score values for each numerical label. The model which gives the best output is Multiclass Decision Forest.

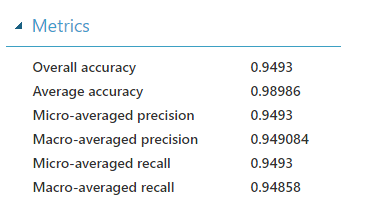
**Evaluation**:

We will use 60000 withheld images in the test set to evaluate our method. The 10000 test images were pre-processed in the same way as those 60000 training images rescaled into size 28X28 and a class label. We won’t touch this test data unless the classifiers were developed. We will apply the trained classifiers on those 60000 withheld images and compare the predicted class label with the true class label. Finally, we mapped the predicted class label to the class name so that our pipeline could serve an application to recognize real-world handwritten digit recognition.

# Technical Approach – Azure Model



# Evaluation Results – Azure Model

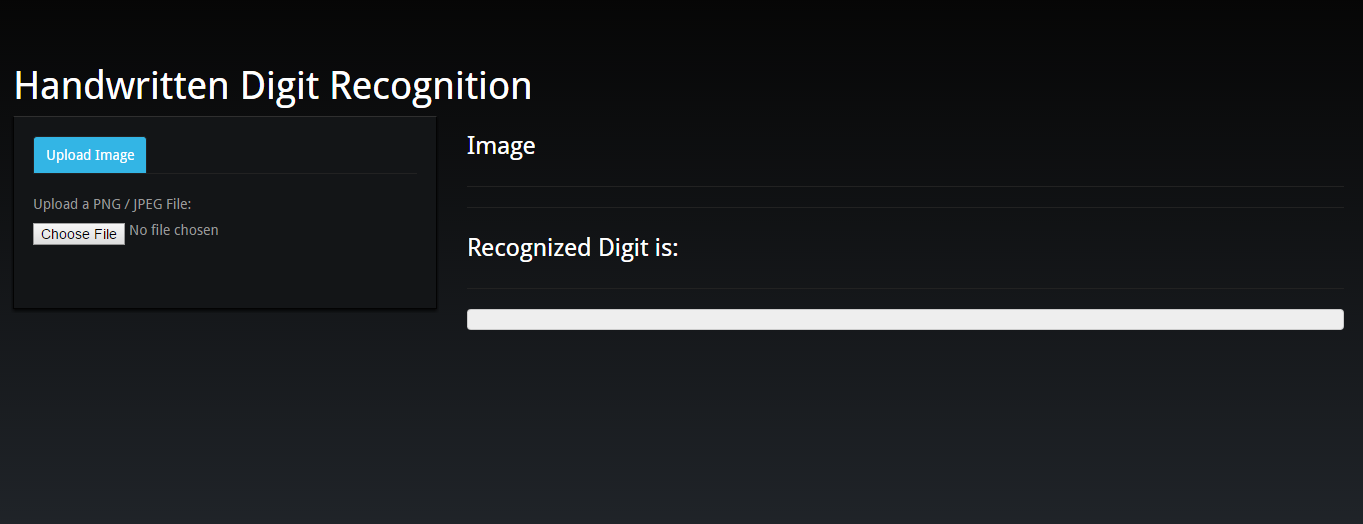


# Evaluation Results on excel -- Label = 2

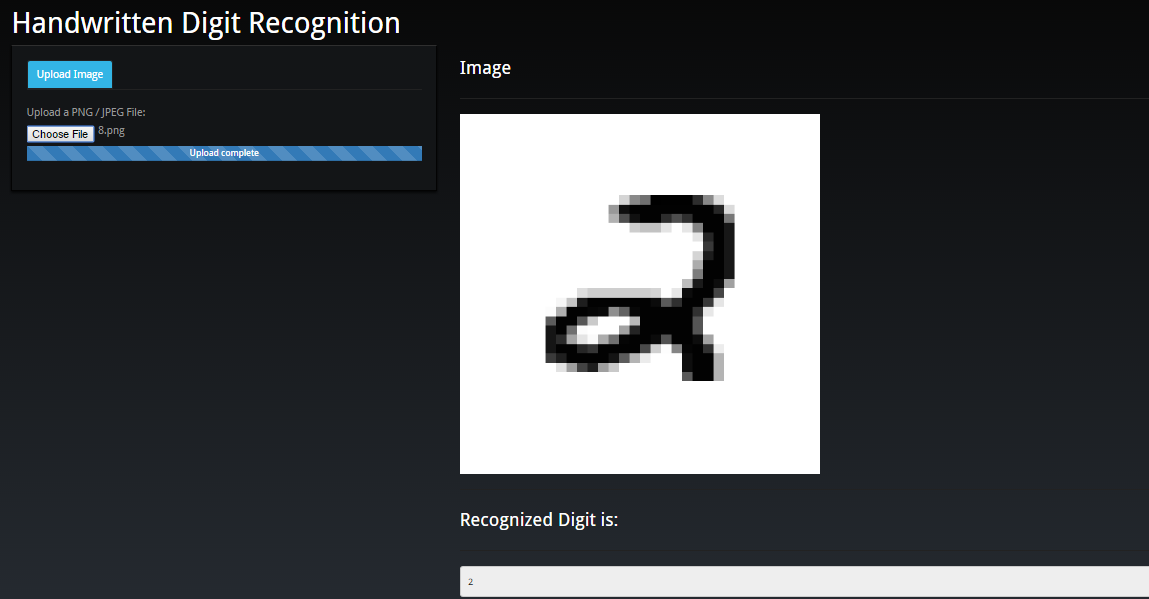
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
| Scored Probabilities for Class "0" | Scored Probabilities for Class "1" | Scored Probabilities for Class "2" | Scored Probabilities for Class "3" | Scored Probabilities for Class "4" | Scored Probabilities for Class "5" | Scored Probabilities for Class "6" | Scored Probabilities for Class "7" | Scored Probabilities for Class "8" | Scored Probabilities for Class "9" | Scored Labels |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **2** |
|  |  |  |  |  |  |  |  |  |  |

# Flow of Digit Recognition on Shiny Application:-

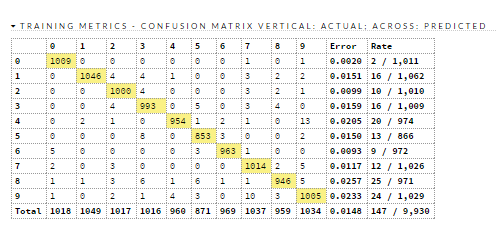
1. Choose the image to be uploaded using the “Choose File” button

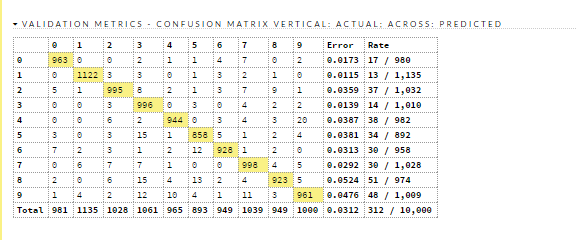
****

1. Based on the scored probabilities obtained from the output of Decision Forest algorithm, it predicts the digit.

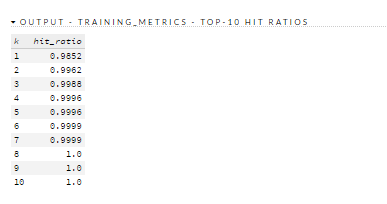
****

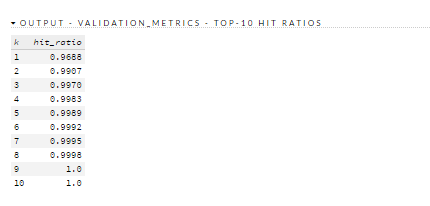
**Evaluation Results on H2O:-**



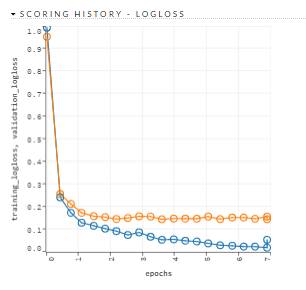


**Hit Ratios:-**





**LogLoss**



**MSE:-**

