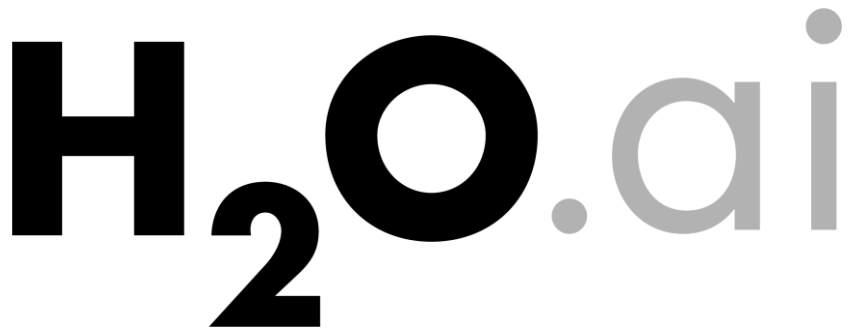
Deep Learning with H_2O



Jo-fai (Joe) Chow

Data Scientist

joe@h2o.ai

@matlabulous

ODSC Masterclass Summit

1st March, 2017

About Me

- Civil (Water) Engineer

  2010 – 2015

  - Consultant (UK)
    - Utilities
    - Asset Management
    - Constrained Optimization
  - Industrial PhD (UK)
    - Infrastructure Design Optimization
    - Machine Learning + Water Engineering
    - Discovered H_2O in 2014

- Data Scientist

  2015

  - Virgin Media (UK)
  - Domino Data Lab (Silicon Valley)

  2016 – Present

  - H_2O.ai (Silicon Valley)

Learning Objectives

- Understand the basic usage of H2O deep learning.
- Use H_2O deep autoencoder for outlier detection.

- (Optional) Introduction to Deep Water
  - Next-Gen H_2O Deep Learning

H_2O Examples

bit.ly/odsc_h2o_dl_2017

woobe / odsc_h2o_deep_learning

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Materials for ODSC Masterclass Summit H2O Deep Learning Tutorials

Edit

New Add topics

8 commits 1 branch 0 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

woobe Updated details Latest commit 06d5031 8 hours ago

LICENSE	Initial commit	2 days ago
MNIST_example.R	MNIST example	8 hours ago
MNIST_example.Rmd	MNIST example	8 hours ago
MNIST_example.html	MNIST example	8 hours ago
README.md	Updated details	8 hours ago
kaggle_mnist_train.csv	MNIST example	8 hours ago
outlier_detection.R	Outlier example	8 hours ago
outlier_detection.Rmd	Outlier example	8 hours ago
outlier_detection.html	Outlier example	8 hours ago

README.md

H2O Deep Learning Tutorials at ODSC Masterclass Summit 2017

About H_2O.ai

Company Overview

Founded	2011 Venture-backed, debuted in 2012
Products	• H_2O Open Source In-Memory AI Prediction Engine • Sparkling Water • Steam
Mission	Operationalize Data Science, and provide a platform for users to build beautiful data products
Team	70 employees • Distributed Systems Engineers doing Machine Learning • World-class visualization designers
Headquarters	Mountain View, CA




Algorithms Overview

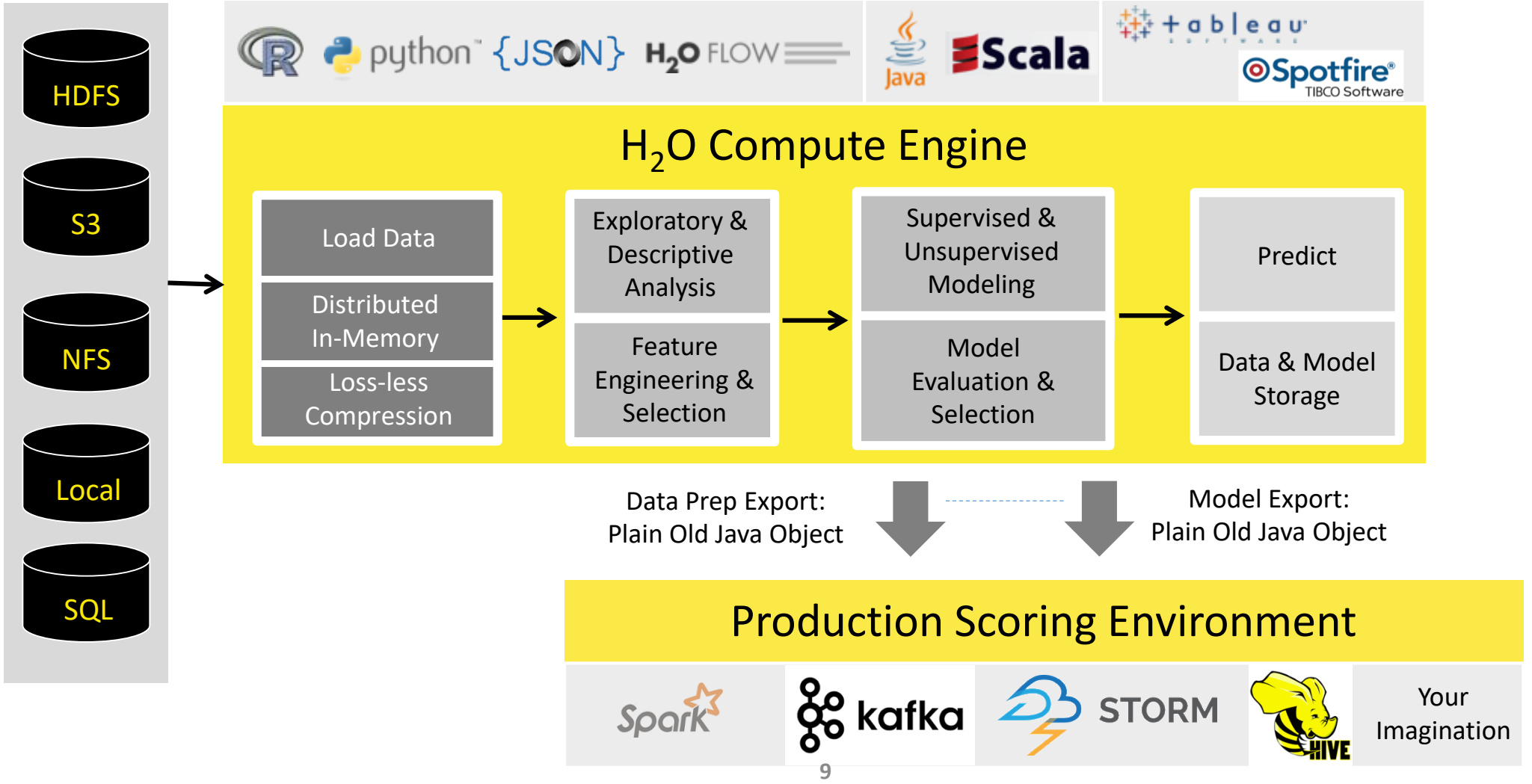
Supervised Learning

Statistical Analysis	• **Generalized Linear Models**: Binomial, Gaussian, Gamma, Poisson and Tweedie • **Naïve Bayes**
Ensembles	• **Distributed Random Forest**: Classification or regression models • **Gradient Boosting Machine**: Produces an ensemble of decision trees with increasing refined approximations
Deep Neural Networks	• **Deep learning**: Create multi-layer feed forward neural networks starting with an input layer followed by multiple layers of nonlinear transformations

Unsupervised Learning

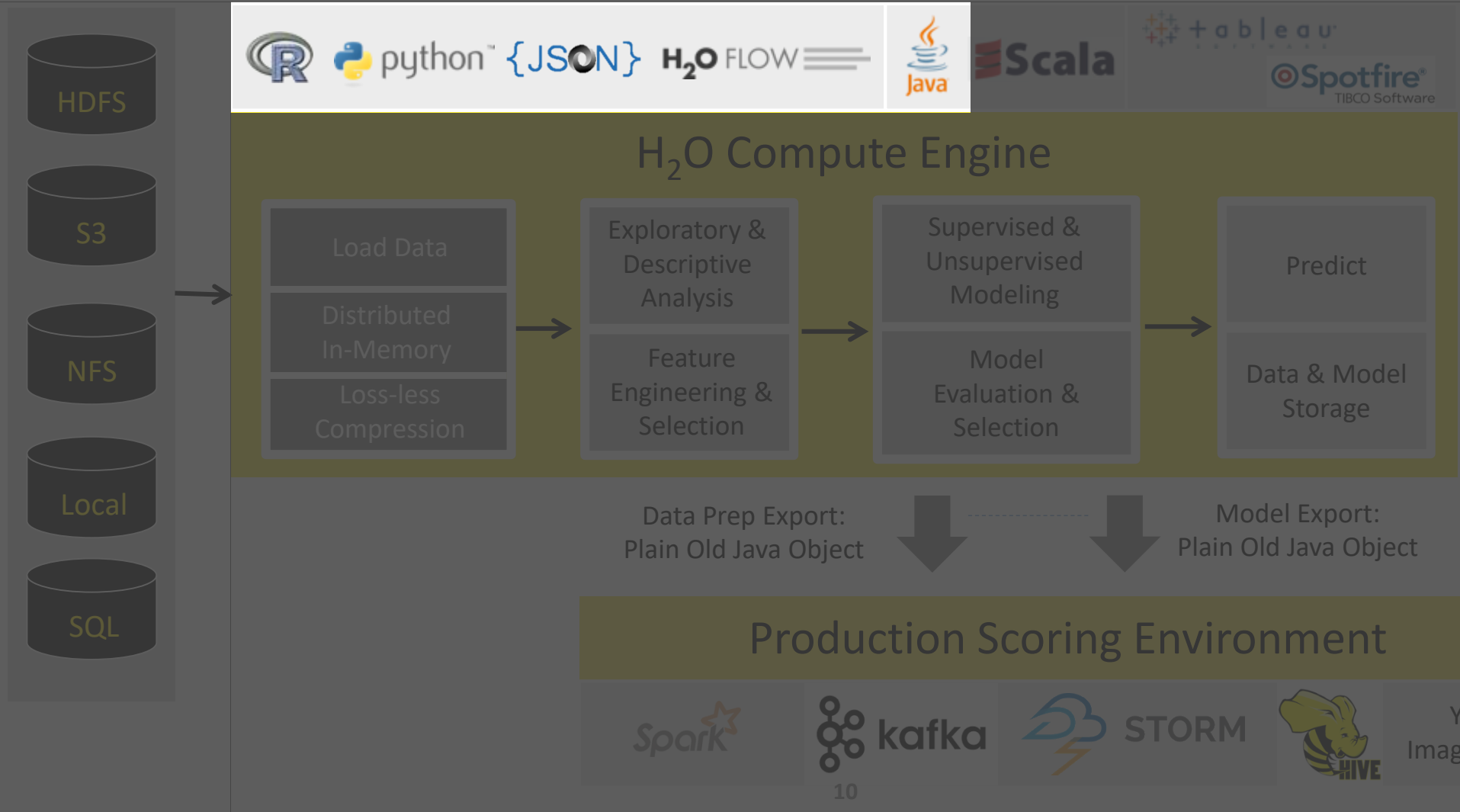
Clustering	• **K-means**: Partitions observations into k clusters/groups of the same spatial size. Automatically detect optimal k
Dimensionality Reduction	• **Principal Component Analysis**: Linearly transforms correlated variables to independent components • **Generalized Low Rank Models:** extend the idea of PCA to handle arbitrary data consisting of numerical, Boolean, categorical, and missing data
Anomaly Detection	• **Autoencoders**: Find outliers using a nonlinear dimensionality reduction using deep learning

High Level Architecture



High Level Architecture

Flow (Web), R, Python API
Java for computation



docs.h2o.ai

Languages

R

- Quick Start Video - R
- R Package Docs
- R Booklet
- Examples and Demos
- R FAQ
- Ensemble R Package Readme
- RSparkling Readme
- Migrating from H2O-2

Python

- Quick Start Video - Python
- Python Module Docs
- Python Booklet
- Examples and Demos
- Python FAQ
- PySparkling Readme 2.0 | 1.6
- skutil Docs

Java

- POJO and MOJO Model Javadoc
- H2O Core Javadoc
- H2O Algorithms Javadoc

Scala

Sparkling Water API	2.0	1.6
Sparkling Water Scaladoc	2.0	1.6
H2O Scaladoc	2.11	2.10

Tutorials, Examples, & Presentations

Tutorials and Blogs

- H2O Tutorials HTML | PDF
- H2O Blogs
- H2O University

Use Case Examples

Chicago crime prediction	R	Python	ScalaSW	PySW
Airlines delays prediction	R	Python	ScalaSW	PySW
Lending Club loan prediction	R	Python	ScalaSW	PySW
Ham or Spam	R	Python	ScalaSW	PySW
Prediction with prostate dataset	R	Python	ScalaSW	PySW

Presentations

- H2O Meetups
- H2O World 2014 Videos
- H2O World 2015 Videos
- Open Tour Chicago Videos
- Open Tour NYC Videos
- Open Tour Dallas Videos

Classic H_2O Deep Learning

Classic Feedforward Neural Network

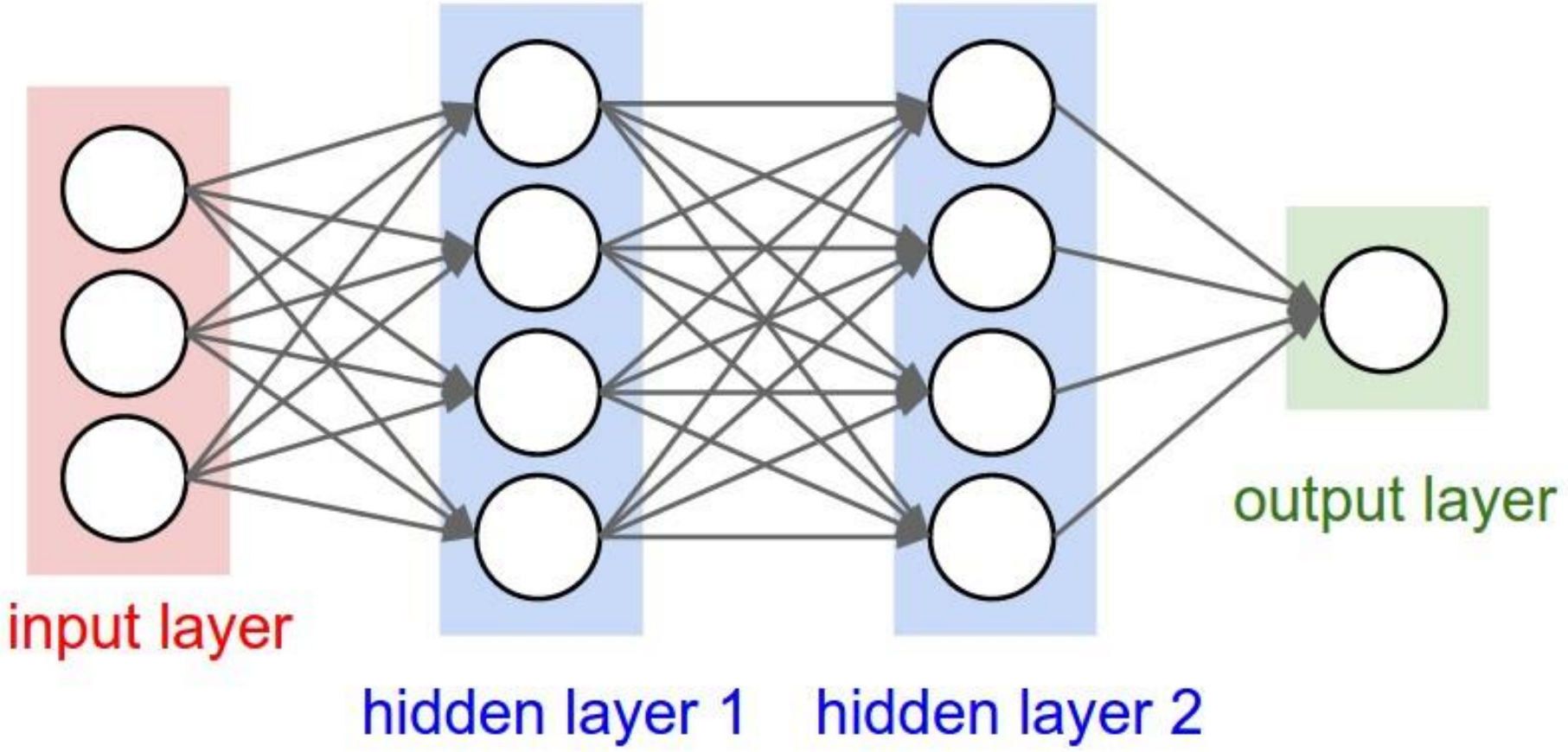


Image credit: http://cs231n.github.io/

H2O.ai

H_2O Deep Water in Action

116M rows, 6GB CSV file
800+ predictors (numeric + categorical)

airlines_all_selected_cols.hex

Actions: View Data | Split... | Build Model... | Predict | Download | Export

Rows	Columns	Compressed Size
116695259	12	2GB

Job

Run Time 00:00:36.712
Remaining Time 00:00:17.188
Type Model
Key deeplearning-dd2f42f7-81f7-42e8-9d98-e34437309828
Description DeepLearning
Status RUNNING
Progress 69%

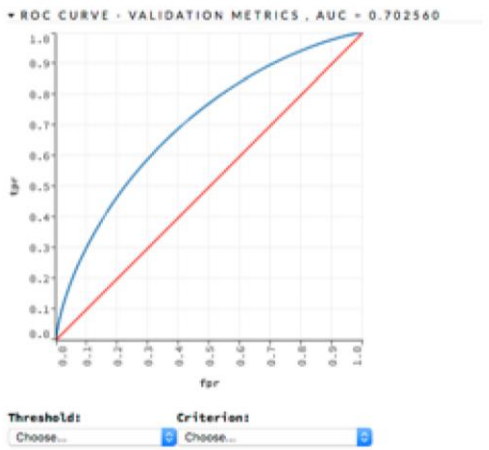
Iterations: 12. Epochs: 0.628821. Speed: 2,243,735 samples/sec. Estimated time left: 21.849 sec

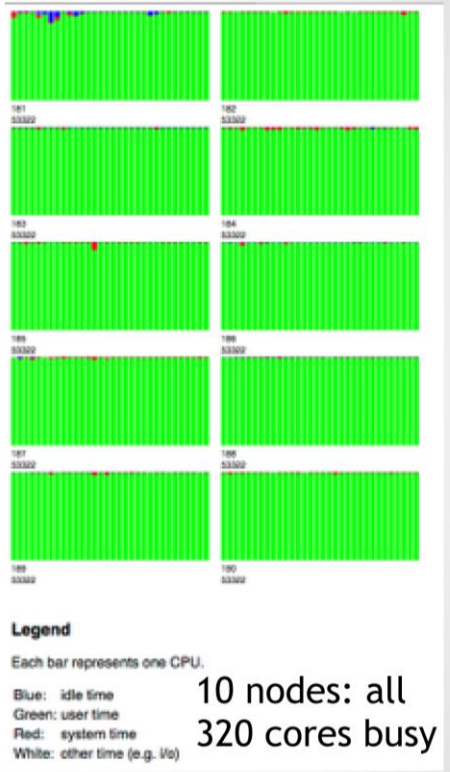
Actions: View | Cancel Job

model trained in <1 min:
2M+ samples/second

▾ OUTPUT - STATUS OF NEURON LAYERS (PREDICTING ISDEPDELAYED, 2-CLASS CLASSIFICATION, BERNOULLI DISTRIBUTION, CROSSENTROPY LOSS, 17,462 WEIGHTS/BIASES, 221.3 KB, 106,585,365 TRAINING SAMPLES, MINI-BATCH SIZE 1)

layer	units	type	dropout	l1	l2	mean_rate	rate_RMS	momentum	mean_weight	weight_RMS	mean_bias	bias_RMS
1	807	Input	0									
2	20	Rectifier	0	0	0	0.0483	0.2020	0	-0.0021	0.2111	-0.9139	1.0036
3	20	Rectifier	0	0	0	0.0197	0.0227	0	-0.1033	0.5362	-1.3988	1.5259
4	20	Rectifier	0	0	0	0.0517	0.0446	0	-0.1575	0.3068	-0.6846	0.6846
5	20	Rectifier	0	0	0	0.0761	0.0844	0	-0.0374	0.2275	-0.2647	0.2481
6	2	Softmax		0	0	0.0161	0.0083	0	0.0741	0.7260	0.4269	0.3056





10 nodes: all
320 cores busy

real-time, interactive
model inspection in Flow

H_2O.ai **Deep Learning Model**

H_2O.ai

H_2O Deep Water in Action

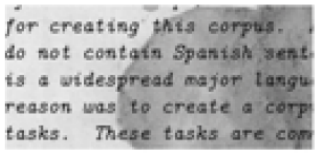
CIFAR-10 Competition Winners: Interviews with Dr. Ben Graham, Phil Culliton, & Zygmunt Zając

Triskelion | 01.02.2015

READ MORE

"I did really like H2O's deep learning implementation in R, though - the interface was great, the back end extremely easy to understand, and it was scalable and flexible. Definitely a tool I'll be going back to."

Kaggle challenge 2nd place winner Colin Priest



Completed • Knowledge • 161 teams

Denoising Dirty Documents

Mon 1 Jun 2015 – Mon 5 Oct 2015 (3 months ago)

READ MORE

"For my final competition submission I used an ensemble of models, including 3 deep learning models built with R and h2o."

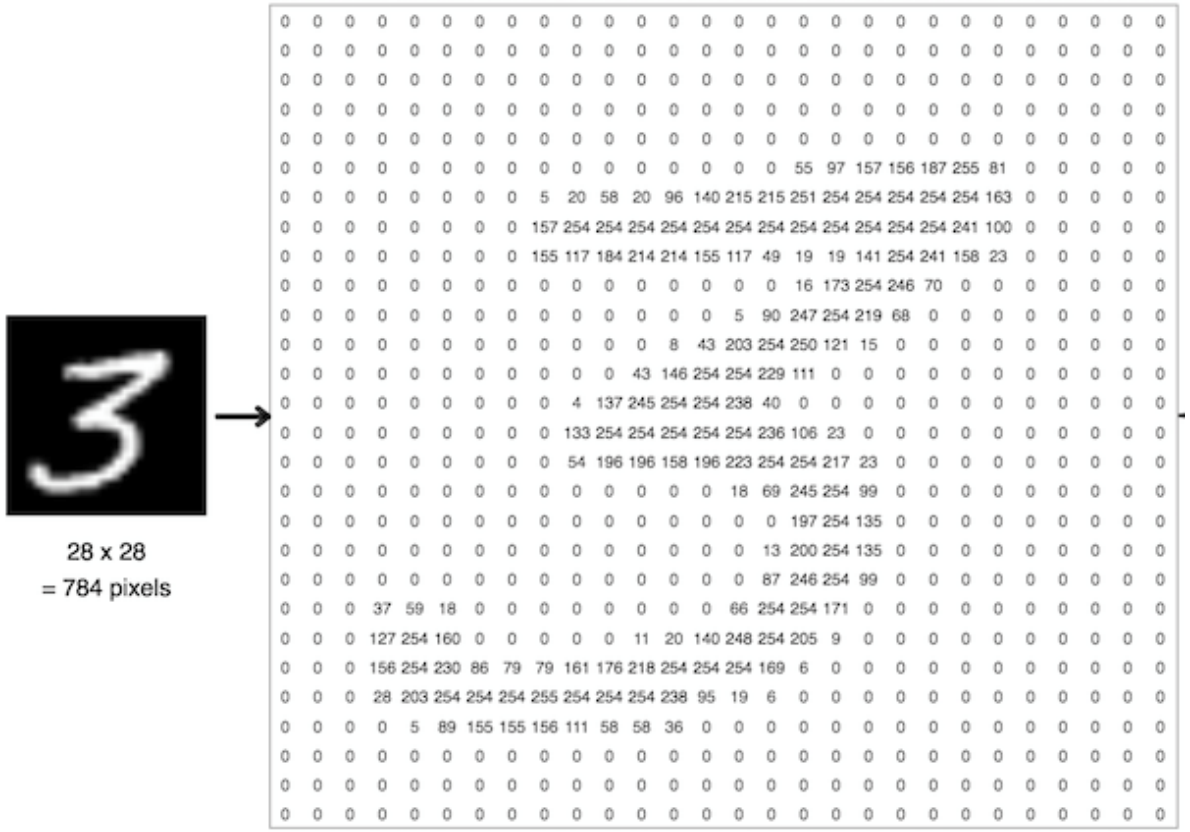
MNIST Example

The Famous Hand-Written Digits Dataset

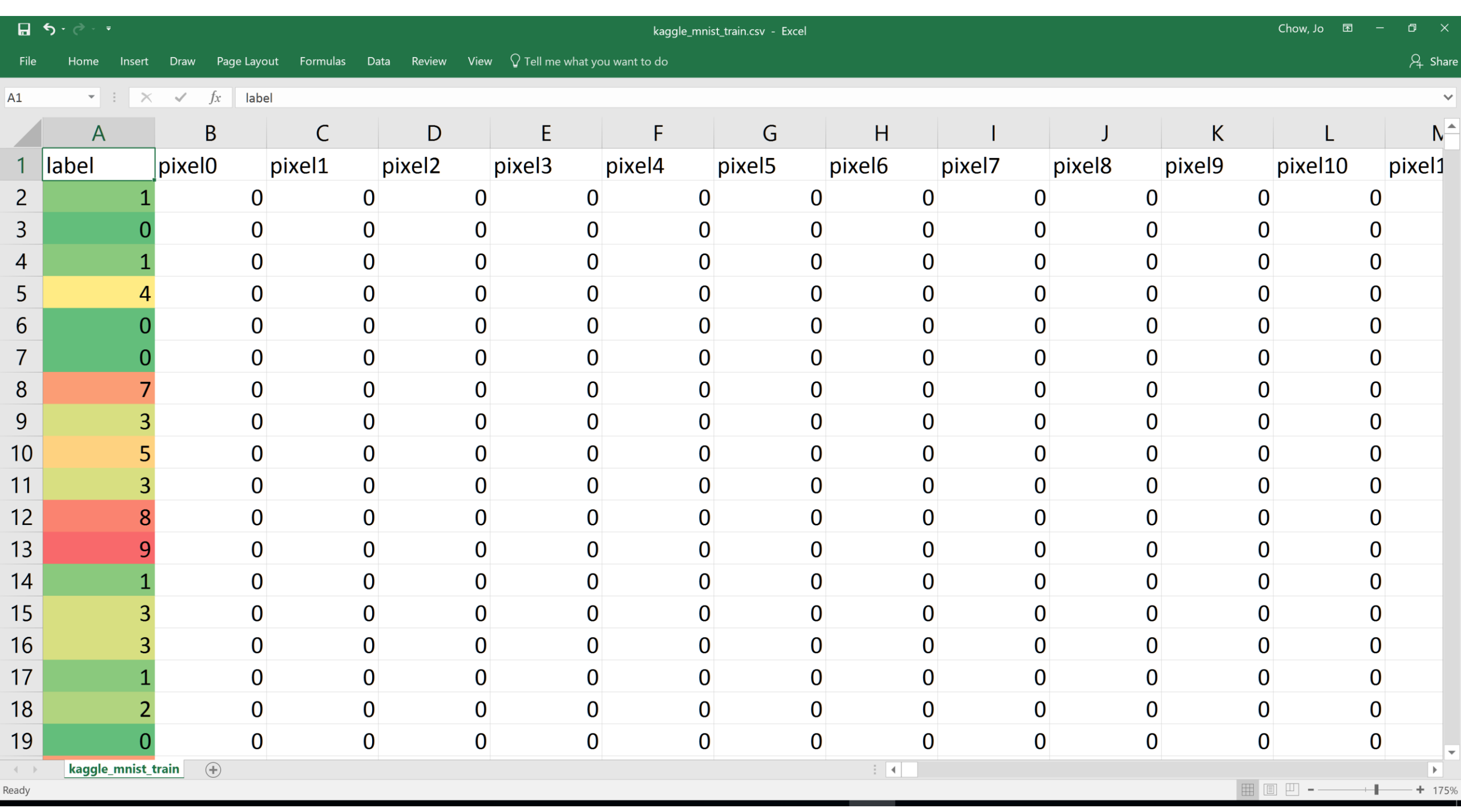
MNIST

MNIST Hand-Written Digits Dataset

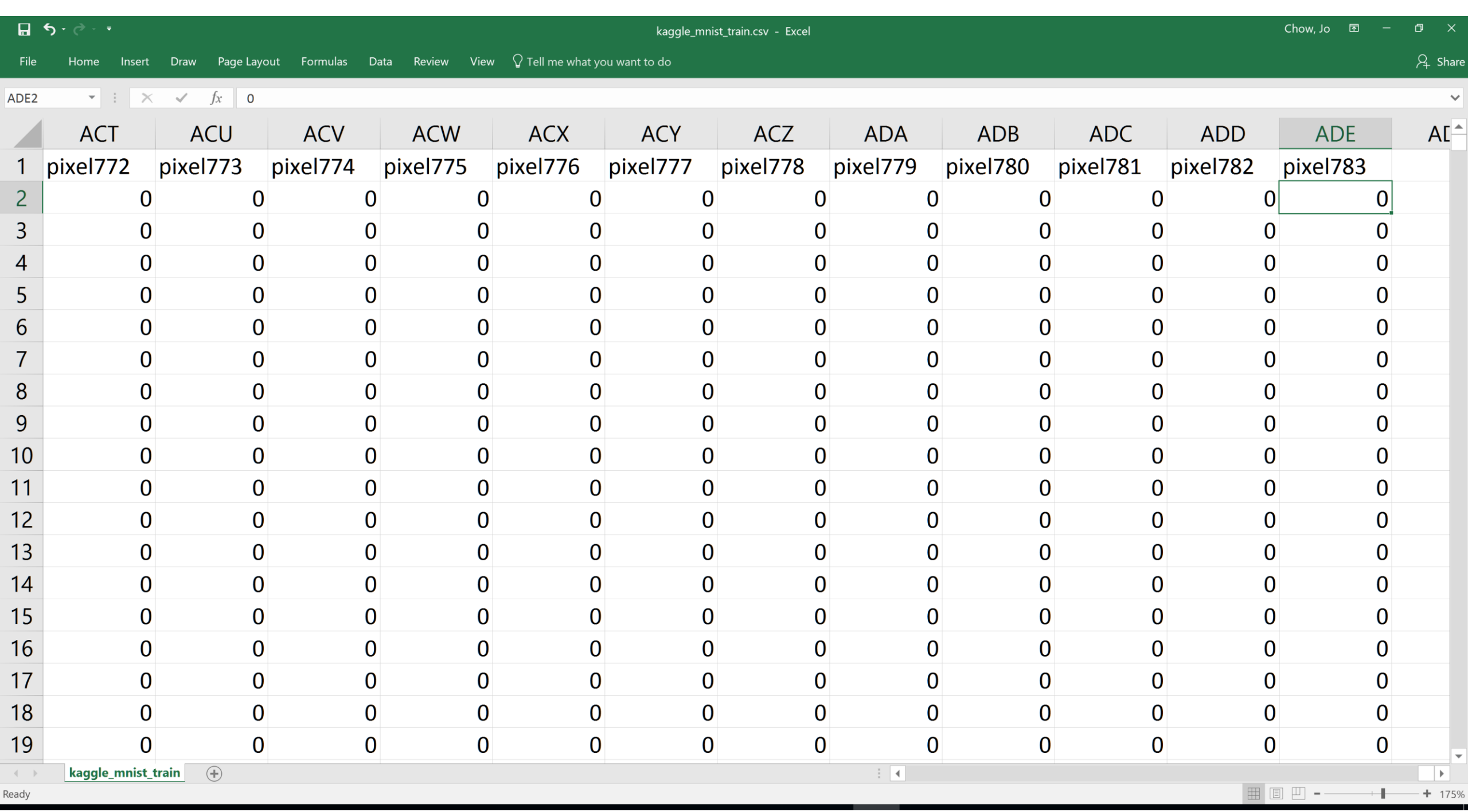
- Inputs
  - 28 x 28 = 784 pixels
  - Range: 0 to 255
    - 0 = Black
    - 1 – 254 = Dark Grey … Light Grey
    - 255 = White
- Output
  - Label: 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9
- CSV File
  - 42,000 Samples (from Kaggle)
  - https://www.kaggle.com/c/digit-recognizer/data



H2O.ai



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel1
2	1	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	0	0	
5	4	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	0	0	
8	7	0	0	0	0	0	0	0	0	0	0	0	
9	3	0	0	0	0	0	0	0	0	0	0	0	
10	5	0	0	0	0	0	0	0	0	0	0	0	
11	3	0	0	0	0	0	0	0	0	0	0	0	
12	8	0	0	0	0	0	0	0	0	0	0	0	
13	9	0	0	0	0	0	0	0	0	0	0	0	
14	1	0	0	0	0	0	0	0	0	0	0	0	
15	3	0	0	0	0	0	0	0	0	0	0	0	
16	3	0	0	0	0	0	0	0	0	0	0	0	
17	1	0	0	0	0	0	0	0	0	0	0	0	
18	2	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0	0	



	ACT	ACU	ACV	ACW	ACX	ACY	ACZ	ADA	ADB	ADC	ADD	ADE
1	pixel772	pixel773	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0

Classic H2O DL - MNIST Example

Jo-fai Chow

1 Mar 2017

R Script: ***MNIST_example.R***

```
# Load R Packages
suppressPackageStartupMessages(library(h2o))

# Start and connect to a local H2O cluster
h2o.init(nthreads = -1)
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /tmp/RtmpU3X4Ca/h2o_joe_started_from_r.out
##     /tmp/RtmpU3X4Ca/h2o_joe_started_from_r.err
##
##
## Starting H2O JVM and connecting: .. Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 seconds 796 milliseconds
##     H2O cluster version:        3.10.3.5
##     H2O cluster version age:    10 days
##     H2O cluster name:           H2O_started_from_R_joe_vfa945
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   5.21 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     R Version:                  R version 3.3.2 (2016-10-31)
```

```
h2o.no_progress()
```

```r
# Import Kaggle MNIST (train)
h_mnist <- h2o.importFile("kaggle_mnist_train.csv")

# Convert label to categorical values
h_mnist$label <- as.factor(h_mnist$label)

# Quick summary
h2o.describe(h_mnist$label)
```

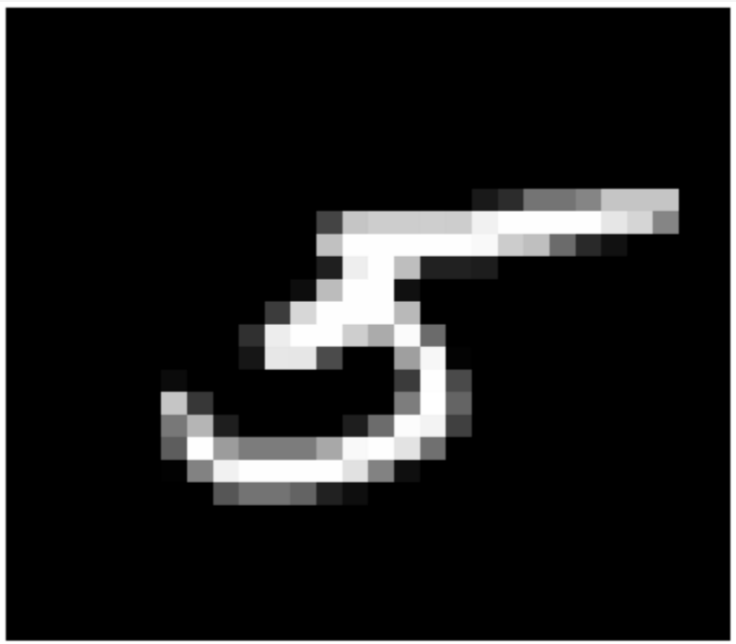
```
##   Label Type Missing Zeros PosInf NegInf Min Max Mean Sigma Cardinality
## 1 label enum       0  4132      0      0   0   9 <NA>  <NA>          10
```

```r
# Define target (y) and features (x)
target <- "label"
features <- setdiff(colnames(h_mnist), target)
print(features)
```

```
##   [1] "pixel0"   "pixel1"   "pixel2"   "pixel3"   "pixel4"   "pixel5"
##   [7] "pixel6"   "pixel7"   "pixel8"   "pixel9"   "pixel10"  "pixel11"
##  [13] "pixel12"  "pixel13"  "pixel14"  "pixel15"  "pixel16"  "pixel17"
##  [19] "pixel18"  "pixel19"  "pixel20"  "pixel21"  "pixel22"  "pixel23"
##  [25] "pixel24"  "pixel25"  "pixel26"  "pixel27"  "pixel28"  "pixel29"
##  [31] "pixel30"  "pixel31"  "pixel32"  "pixel33"  "pixel34"  "pixel35"
##  [37] "pixel36"  "pixel37"  "pixel38"  "pixel39"  "pixel40"  "pixel41"
##  [43] "pixel42"  "pixel43"  "pixel44"  "pixel45"  "pixel46"  "pixel47"
##  [49] "pixel48"  "pixel49"  "pixel50"  "pixel51"  "pixel52"  "pixel53"
##  [55] "pixel54"  "pixel55"  "pixel56"  "pixel57"  "pixel58"  "pixel59"
##  [61] "pixel60"  "pixel61"  "pixel62"  "pixel63"  "pixel64"  "pixel65"
##  [67] "pixel66"  "pixel67"  "pixel68"  "pixel69"  "pixel70"  "pixel71"
##  [73] "pixel72"  "pixel73"  "pixel74"  "pixel75"  "pixel76"  "pixel77"
##  [79] "pixel78"  "pixel79"  "pixel80"  "pixel81"  "pixel82"  "pixel83"
##  [85] "pixel84"  "pixel85"  "pixel86"  "pixel87"  "pixel88"  "pixel89"
##  [91] "pixel90"  "pixel91"  "pixel92"  "pixel93"  "pixel94"  "pixel95"
##  [97] "pixel96"  "pixel97"  "pixel98"  "pixel99"  "pixel100" "pixel101"
## [103] "pixel102" "pixel103" "pixel104" "pixel105" "pixel106" "pixel107"
## [109] "pixel108" "pixel109" "pixel110" "pixel111" "pixel112" "pixel113"
## [115] "pixel114" "pixel115" "pixel116" "pixel117" "pixel118" "pixel119"
## [121] "pixel120" "pixel121" "pixel122" "pixel123" "pixel124" "pixel125"
## [127] "pixel126" "pixel127" "pixel128" "pixel129" "pixel130" "pixel131"
## [133] "pixel132" "pixel133" "pixel134" "pixel135" "pixel136" "pixel137"
## [139] "pixel138" "pixel139" "pixel140" "pixel141" "pixel142" "pixel143"
```

R Script: *MNIST_example.R*

```r
# Custom function to visualise digit
show_digit <- function(h_frame, features) {

  # Convert to normal R data frame
  d <- as.data.frame(h_frame[, features])

  # Reshape
  m <- matrix(data = as.numeric(d), nrow = 28, ncol = 28, byrow = TRUE)
  rotate <- function(x) t(apply(x, 2, rev))
  m <- rotate(m)

  # Show image
  image(m, axes = FALSE, col = grey(seq(0, 1, length = 256)))

}

# Test
show_digit(h_mnist[sample(1:100, 1),], features)
```



R Script: ***MNIST_example.R***

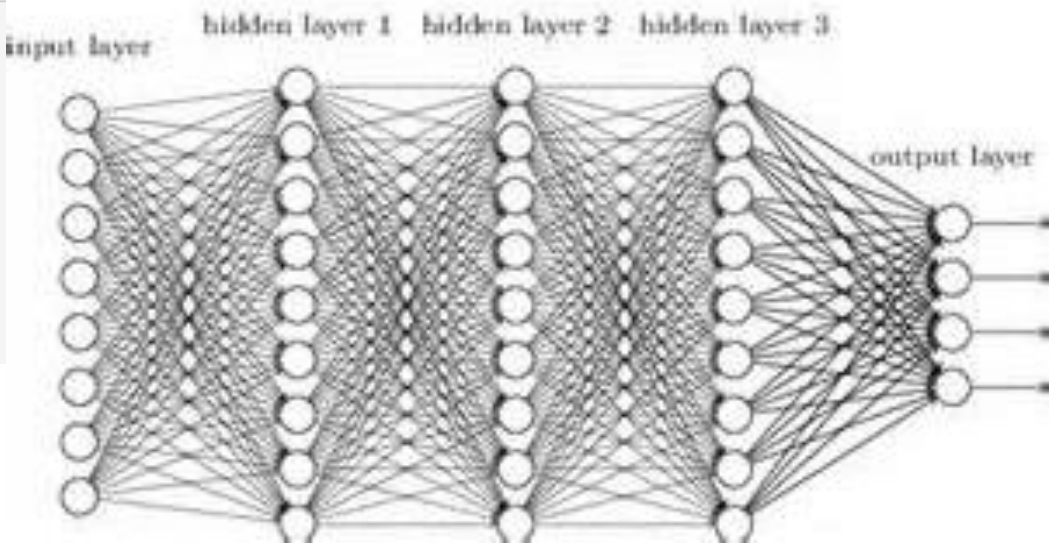
```
# Split dataset into train/test
h_split <- h2o.splitFrame(h_mnist, ratios = 0.8, seed = 1234)
h_train <- h_split[[1]]
h_test <- h_split[[2]]
```

R Script: ***MNIST_example.R***

```
# Build a Classic H2O Deep Learning Model with Manual Settings
model_manual <- h2o.deeplearning(x = features,
                                 y = target,
                                 training_frame = h_train,
                                 activation = "Rectifier",
                                 hidden = c(100, 100, 100),
                                 epochs = 50)
```

Layer 1:
100 nodes

Layer 3:
100 nodes



Inputs:
784

Layer 2:
100 nodes

Outputs:
10

```r
# Evaluate
h2o.performance(model_manual, newdata = h_test)
```

```
## H2OMultinomialMetrics: deeplearning
##
## Test Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.02683314
## RMSE: (Extract with `h2o.rmse`) 0.1638083
## Logloss: (Extract with `h2o.logloss`) 0.2830743
## Mean Per-Class Error: 0.02948618
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
## ========================================================================
## Confusion Matrix: vertical: actual; across: predicted
##          0   1   2   3   4   5   6   7   8   9  Error          Rate
## 0      824   0   1   0   5   3   1   0   4   1 0.0179 =    15 / 839
## 1        0 913   1   5   2   1   1   3   3   1 0.0183 =    17 / 930
## 2        0   4 775   4   2   1   2  10   7   2 0.0397 =    32 / 807
## 3        1   0  10 835   2  12   0   4   3   3 0.0402 =    35 / 870
## 4        0   1   5   0 791   0   2   0   2   5 0.0186 =    15 / 806
## 5        3   1   1   9   2 702   4   0   6   3 0.0397 =    29 / 731
## 6        3   2   3   0   3   7 810   0   3   0 0.0253 =    21 / 831
## 7        1   3   7   1   5   1   0 859   0   7 0.0283 =    25 / 884
## 8        2   2   1   4   0  10   4   2 762   2 0.0342 =    27 / 789
## 9        0   1   0   8  10   1   0   5   2 797 0.0328 =    27 / 824
## Totals 834 927 804 866 822 738 824 883 792 821 0.0292 = 243 / 8,311
##
```

H2O Deep Learning on MNIST: 0.87% test set error (so far)

Scoring history

Time taken for last scoring: 4.928 sec

Number of training data samples for scoring: 9942

Number of validation data samples for scoring: all 10000

Toggle view of plot of classification error on training data

Toggle view of plot of classification error on validation set

test set error: 1.5% after 10 mins
1.0% after 1.5 hours
0.87% after 4 hours

On 4 nodes

Training Time	Training Epochs	Training Samples	Training Error	Validation Error
4:29:51.686	1925.92	115,555,193	0.00 %	0.95 %
4:28:13.147	1916.92	115,015,339	0.00 %	0.95 %
4:26:23.201	1906.93	114,415,501	0.00 %	0.87 %
4:24:43.292	1897.94	113,876,276	0.00 %	0.95 %
4:23:04.435	1888.93	113,335,946	0.01 %	0.95 %
4:21:25.901	1879.93	112,795,743	0.01 %	1.01 %
4:19:47.285	1870.91	112,254,822	0.02 %	0.94 %

World-class results!
No pre-training
No distortions
No convolutions
No unsupervised training

Model type: Classification, predicting: C785

Number of model parameters (weights/biases): 3,904,522

Progress

Status of Neuron Layers

#	Units	Type	Dropout	L1
1	717	Input	20.00 %	
2	1024	RectifierDropout	50.00 %	1.0E-5
3	1024	RectifierDropout	50.00 %	1.0E-5
4	2048	RectifierDropout	50.00 %	1.0E-5
5	10	Softmax		1.0E-5

Classification error on training data: 0.00 %

Classification error on validation data: 0.98 %

Training samples: 42,000,000

Epochs: 700.000 / 10000.000

Number of compute nodes: 4 (128 threads)

Training samples per iteration: 240,000

Training speed: 7,904 samples/s

Training time: 1:28:33.643

Running on 4 nodes with 16 cores each

Frequent errors: confuse 2/7 and 4/9

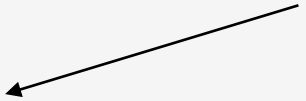
Confusion Matrix

Actual / Predicted	0	1	2	3	4	5	6	7	8	9	Error
0	974	1	1	0	0	0	2	1	1	0	0.00612 = 6 / 980
1	0	1,134	1	0	0	0	0	0	0	0	0.00088 = 1 / 1,135
2	0	0	1,025	1	1	0	0	4	1	0	0.00678 = 7 / 1,032
3	1	0	2	1,000	0	0	0	4	2	1	0.00990 = 10 / 1,010
4	0	0	2	0	973	0	4	0	0	3	0.00916 = 9 / 982
5	2	0	0	4	0	883	1	1	1	0	0.01009 = 9 / 892
6	3	3	0	1	1	1	949	0	0	0	0.00939 = 9 / 958
7	1	2	9	1	0	0	0	1,014	1	0	0.01362 = 14 / 1,028
8	1	1	2	4	0	3	0	3	958	2	0.01643 = 16 / 974
9	2	2	0	2	6	3	0	3	0	991	0.01784 = 18 / 1,009
Totals	984	1,143	1,042	1,013	981	890	956	1,030	964	997	0.00990 = 99 / 10,000

Reproducible Deep Learning Model

R Script: ***MNIST_example.R***

```
# Build a Reproducible Classic H2O Deep Learning Model
# DL with reproducible = TRUE and seed
# Note 1: using one CPU thread only. Can be very slow.
# Note 2: using a small network (50, 50) and one epoch for demo only.
model_repro <- h2o.deeplearning(x = features,
                                y = target,
                                training_frame = h_train,
                                hidden = c(50, 50),
                                epochs = 1,
                                reproducible = TRUE,
                                seed = 1234)
```

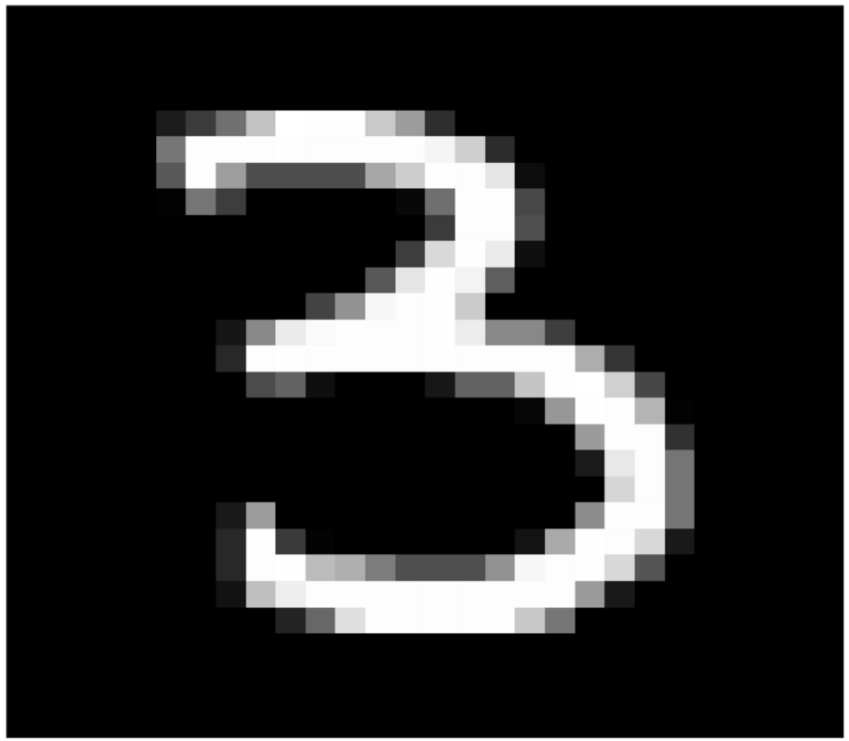


Making Predictions

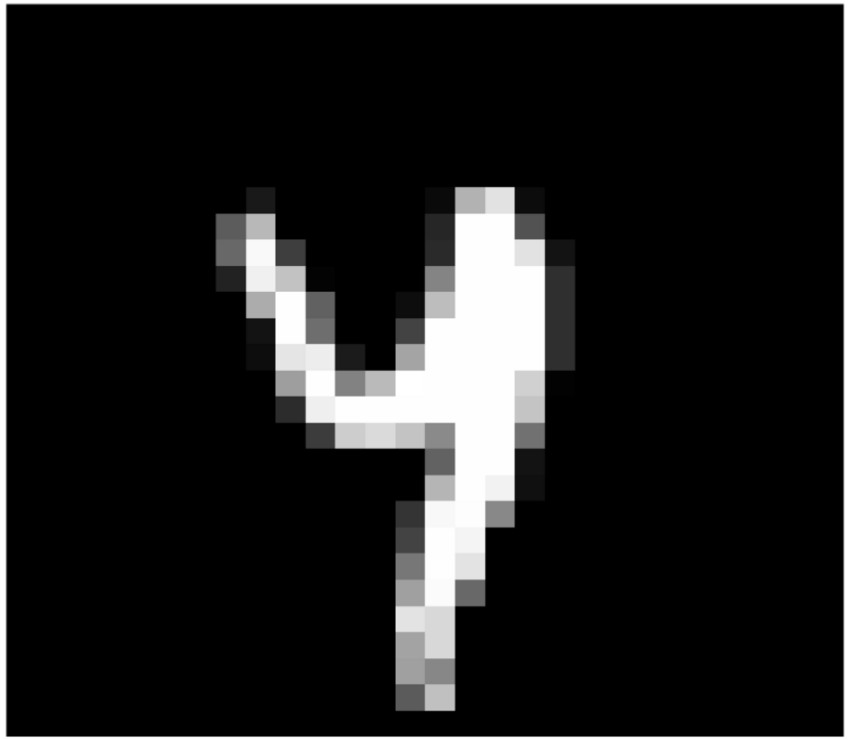
R Script: ***MNIST_example.R***

```
# Make predictions
yhat_test <- h2o.predict(model_manual, newdata = h_test)
head(yhat_test)
```

```
##   predict           p0           p1           p2           p3           p4
## 1       4 5.223289e-13 2.501115e-11 5.265284e-10 2.458560e-15 9.997615e-01
## 2       3 2.007449e-29 2.151090e-24 1.312613e-18 1.000000e+00 7.287101e-39
## 3       3 7.179657e-25 4.004403e-31 3.055718e-18 1.000000e+00 1.759169e-34
## 4       0 1.000000e+00 1.216938e-26 1.655580e-17 3.509741e-24 3.694056e-27
## 5       7 7.111701e-20 8.382516e-13 1.100512e-07 4.805975e-13 2.014590e-28
## 6       5 1.747535e-26 8.429423e-38 2.442106e-25 1.227226e-24 1.706396e-33
##             p5           p6           p7           p8           p9
## 1 5.069687e-13 4.860861e-09 2.850294e-10 3.756945e-11 2.384876e-04
## 2 1.583232e-20 3.281022e-33 8.611700e-21 2.394772e-25 1.538073e-20
## 3 2.768688e-19 6.716212e-28 6.347359e-20 1.268122e-16 5.434987e-13
## 4 6.670455e-19 9.435549e-17 8.288200e-30 1.336547e-25 7.208050e-24
## 5 4.502765e-15 2.115472e-20 9.999999e-01 8.355626e-20 4.396917e-17
## 6 9.999999e-01 1.022763e-07 7.567215e-44 4.901693e-12 4.959423e-28
```



```
## Ground Truth:  3
## Model Prediction:
##   predict           p0           p1           p2 p3           p4
## 1       3 6.698649e-34 2.577556e-20 1.361243e-17  1 2.416728e-32
##             p5           p6           p7           p8           p9
## 1 6.394046e-29 8.989755e-34 2.341015e-29 2.637714e-30 1.99888e-27
##
## [1 row x 11 columns]
```



```
## Ground Truth:  4
## Model Prediction:
##   predict           p0           p1           p2           p3 p4
## 1       4 2.305157e-18 4.232902e-17 1.102188e-22 8.131899e-24  1
##             p5           p6           p7           p8           p9
## 1 3.638836e-25 8.476756e-18 3.629665e-15 1.321357e-18 3.943542e-10
##
## [1 row x 11 columns]
```

Outlier Detection Example

Outlier Detection

- Definition
  - Identification of items, events or observations which do not conform to an expected pattern or other items in a dataset.

- Applications
  - Bank Fraud
  - Monitoring Manufacturing Lines
  - Machine Learning

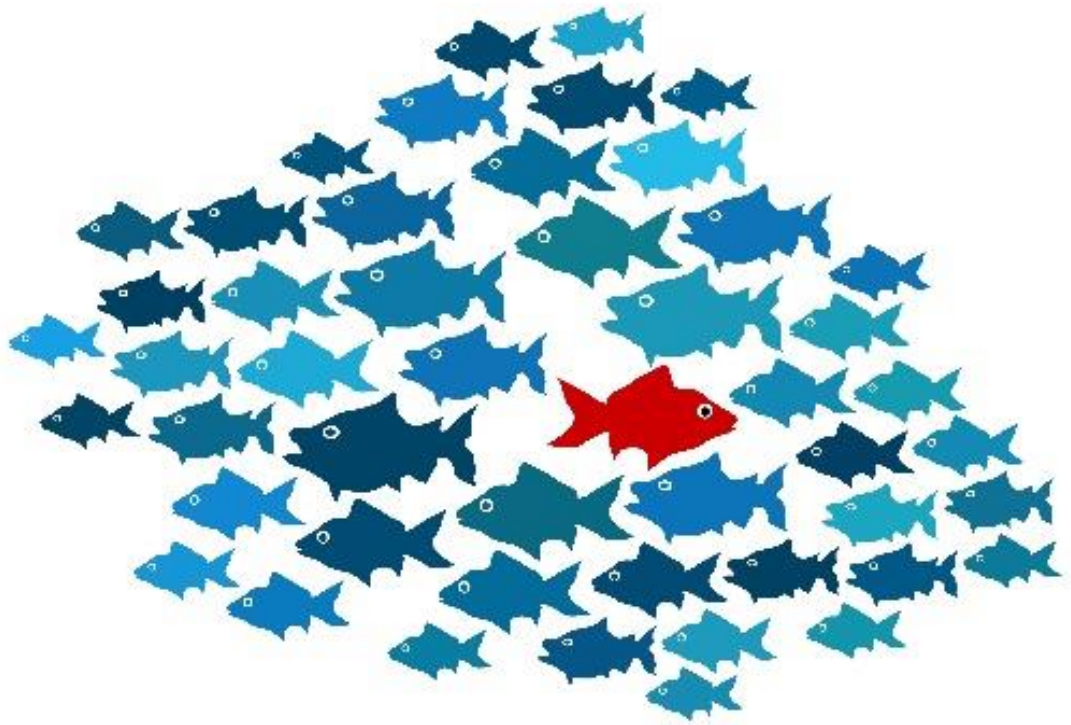
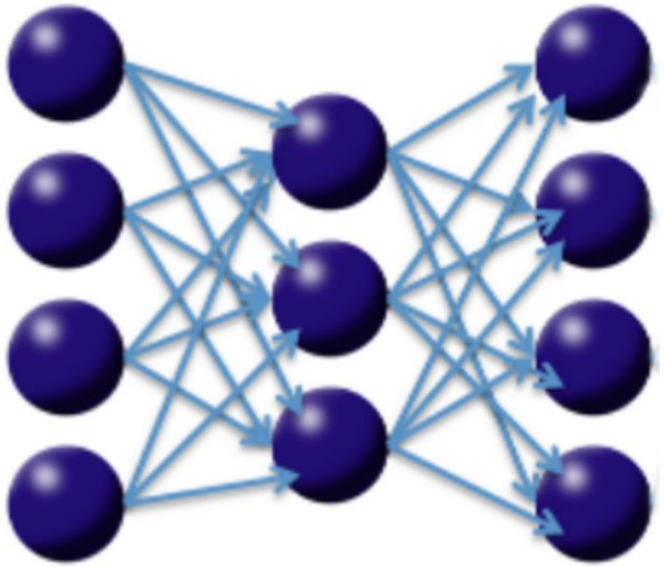


Photo credit: www.dbta.com

Anomaly Detection on MNIST with H2O Deep Learning

This tutorial shows how a Deep Learning Auto-Encoder model can be used to find outliers in a dataset. This file is both valid R and markdown code.

Consider the following three-layer neural network with one hidden layer and the same number of input neurons (features) as output neurons. The loss function is the MSE between the input and the output. Hence, the network is forced to learn the identity via a nonlinear, reduced representation of the original data. Such an algorithm is called a deep autoencoder; these models have been used extensively for unsupervised, layer-wise pretraining of supervised deep learning tasks, but here we consider the autoencoder's application for discovering anomalies in data.



https://github.com/h2oai/h2o-training-book/blob/master/hands-on_training/anomaly_detection.md

We use the well-known MNIST dataset of hand-written digits, where each row contains the 28^2=784 raw gray-scale pixel values from 0 to 255 of the digitized digits (0 to 9).

Finding outliers - ugly hand-written digits

We train a Deep Learning Auto-Encoder to learn a compressed (low-dimensional) non-linear representation of the dataset, hence learning the intrinsic structure of the training dataset. The auto-encoder model is then used to transform all test set images to their reconstructed images, by passing through the lower-dimensional neural network. We then find outliers in a test dataset by comparing the reconstruction of each scanned digit with its original pixel values. The idea is that a high reconstruction error of a digit indicates that the test set point doesn't conform to the structure of the training data and can hence be called an outlier.

1. Learn what's *normal* from the training data

Train unsupervised Deep Learning autoencoder model on the training dataset. For simplicity, we train a model with 1 hidden layer of 50 Tanh neurons to create 50 non-linear features with which to reconstruct the original dataset. We learned from the Dimensionality Reduction tutorial that 50 is a reasonable choice. For simplicity, we train the auto-encoder for only 1 epoch (one pass over the data). We explicitly include constant columns (all white background) for the visualization to be easier.

```
ae_model <- h2o.deeplearning(x=predictors,
                training_frame=train.hex,
                hidden=c(50),
                epoch=1,
                activation="Tanh",
                autoencoder=T,
                ignore_const_cols=F)
```

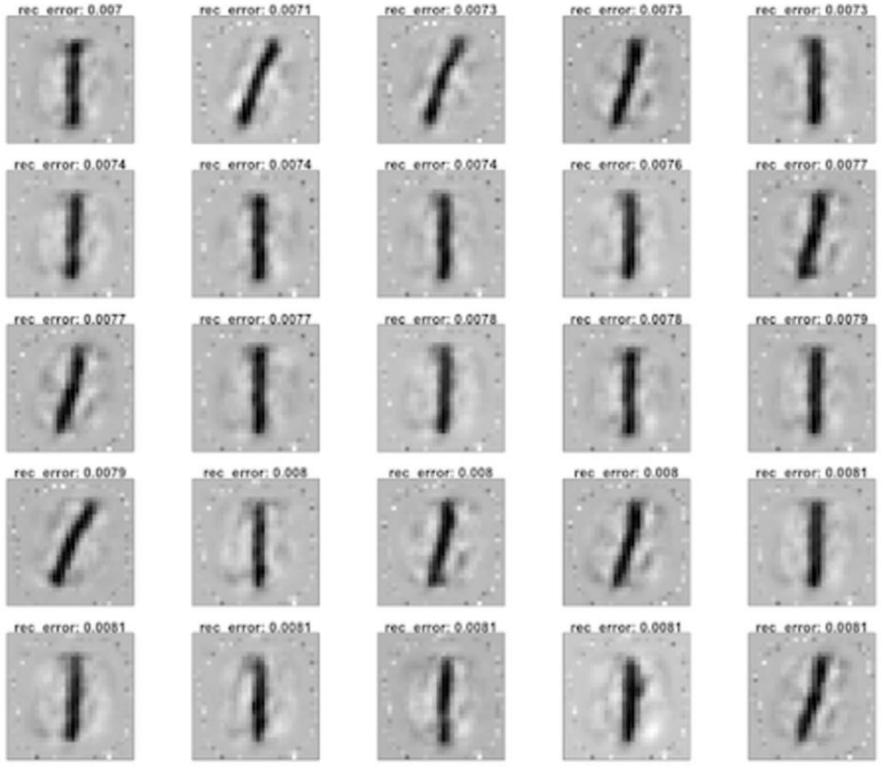
```
summary(test_recon)
```

The good

Let's plot the 25 digits with lowest reconstruction error. First we plot the reconstruction, then the original scanned images.

```
plotDigits(test_recon, test_rec_error, c(1:25))
plotDigits(test_hex,   test_rec_error, c(1:25))
```

Reconstruction



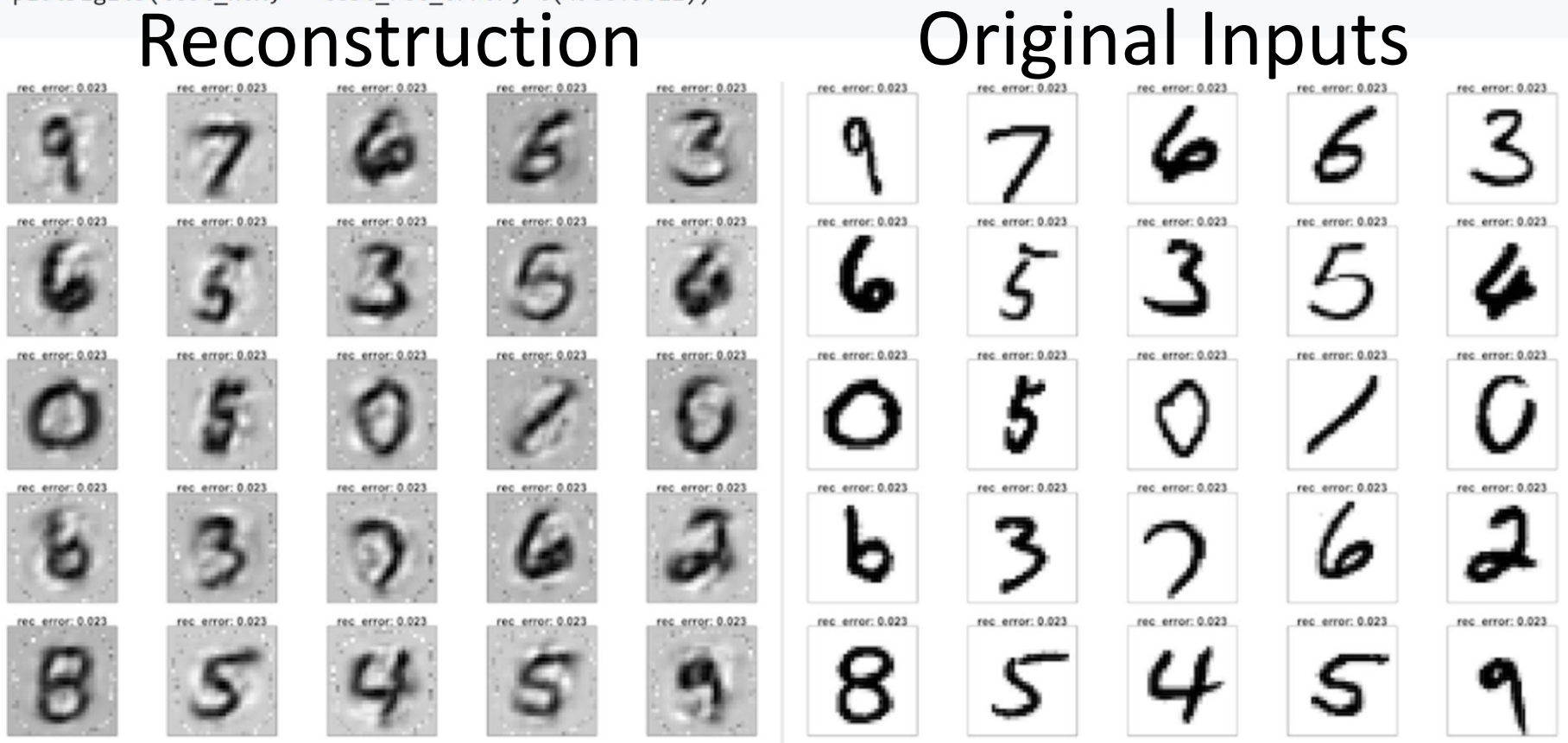
Original Inputs

Clearly, a well-written digit 1 appears in both the training and testing set, and is easy to reconstruct by the autoencoder with minimal reconstruction error. Nothing is as easy as a straight line.

The bad

Now let's look at the 25 digits with median reconstruction error.

```
plotDigits(test_recon, test_rec_error, c(4988:5012))
plotDigits(test_hex,   test_rec_error, c(4988:5012))
```



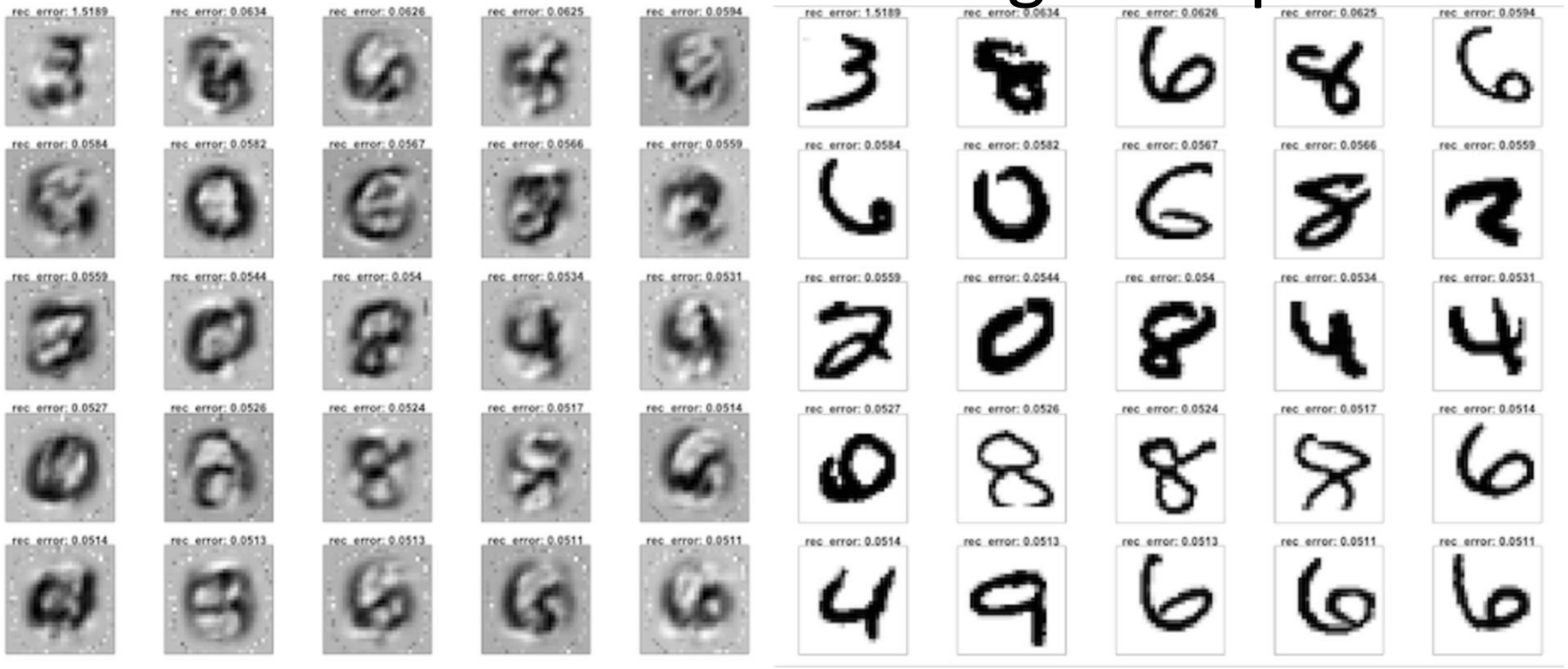
These test set digits look "normal" - it is plausible that they resemble digits from the training data to a large extent, but they do have some particularities that cause some reconstruction error.

The ugly

And here are the biggest outliers - The 25 digits with highest reconstruction error!

```
plotDigits(test_recon, test_rec_error, c(9976:10000))
plotDigits(test_hex,   test_rec_error, c(9976:10000))
```

Reconstruction Original Inputs



Now here are some pretty ugly digits that are plausibly not commonly found in the training data - some are even hard to classify by humans.

Voila!

How about non-image data?

Outlier Detection

Jo-fai Chow

1 March 2017

R Script: ***outlier_detection.R***

```
# Load R Packages
suppressPackageStartupMessages(library(h2o))
suppressPackageStartupMessages(library(mlbench)) # for datasets

# Start and connect to a local H2O cluster
h2o.init(nthreads = -1)
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /tmp/Rtmphx3IP6/h2o_joe_started_from_r.out
##     /tmp/Rtmphx3IP6/h2o_joe_started_from_r.err
##
##
## Starting H2O JVM and connecting: .. Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 seconds 118 milliseconds
##     H2O cluster version:        3.10.3.5
##     H2O cluster version age:    10 days
```

Boston Housing Dataset

UCI Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Housing Data Set

Download: Data Folder, Data Set Description

Abstract: Taken from StatLib library

Data Set Characteristics:	Multivariate	**Number of Instances:**	506	**Area:**	N/A
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	14	**Date Donated**	1993-07-07
Associated Tasks:	Regression	**Missing Values?**	No	**Number of Web Hits:**	274524

Source:

Origin:

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

Creator:

Harrison, D. and Rubinfeld, D.L.
'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

Data Set Information:

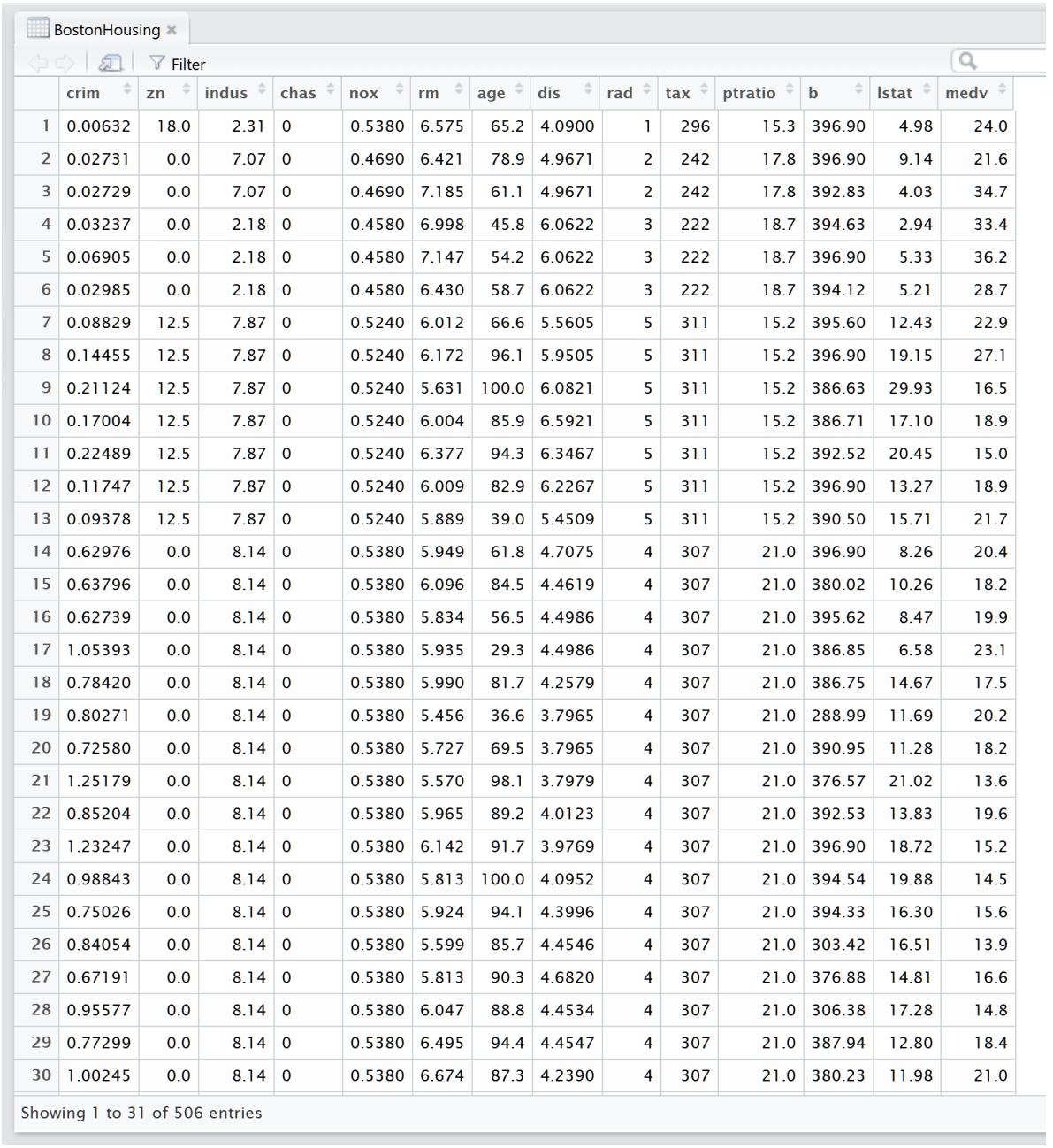
Concerns housing values in suburbs of Boston.

Boston Housing Dataset

Attribute Information:

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per $10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000's

Boston Housing



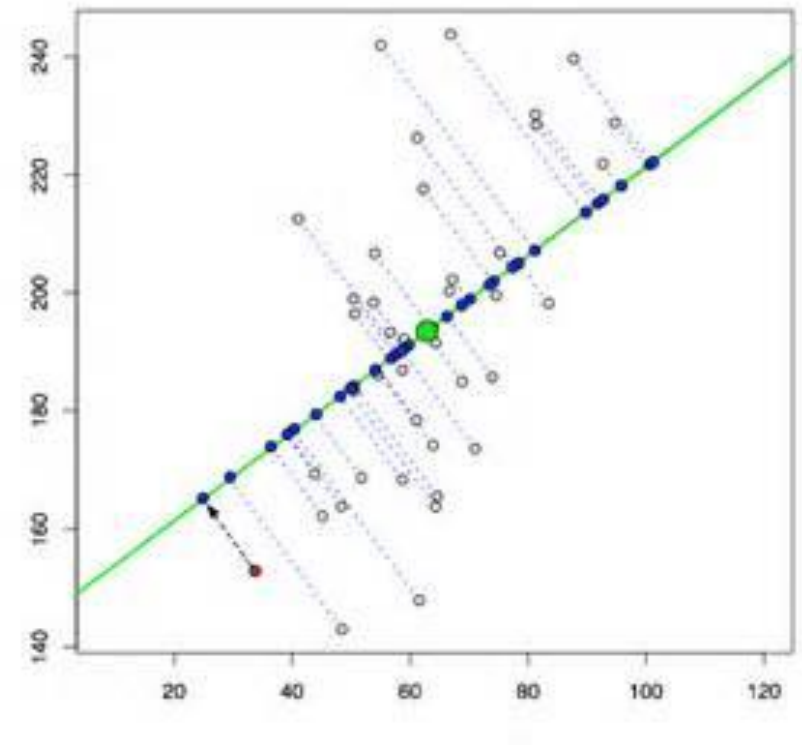
BostonHousing

Filter

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
1	0.00632	18.0	2.31	0	0.5380	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
2	0.02731	0.0	7.07	0	0.4690	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
3	0.02729	0.0	7.07	0	0.4690	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
4	0.03237	0.0	2.18	0	0.4580	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
5	0.06905	0.0	2.18	0	0.4580	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
6	0.02985	0.0	2.18	0	0.4580	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
7	0.08829	12.5	7.87	0	0.5240	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	22.9
8	0.14455	12.5	7.87	0	0.5240	6.172	96.1	5.9505	5	311	15.2	396.90	19.15	27.1
9	0.21124	12.5	7.87	0	0.5240	5.631	100.0	6.0821	5	311	15.2	386.63	29.93	16.5
10	0.17004	12.5	7.87	0	0.5240	6.004	85.9	6.5921	5	311	15.2	386.71	17.10	18.9
11	0.22489	12.5	7.87	0	0.5240	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15.0
12	0.11747	12.5	7.87	0	0.5240	6.009	82.9	6.2267	5	311	15.2	396.90	13.27	18.9
13	0.09378	12.5	7.87	0	0.5240	5.889	39.0	5.4509	5	311	15.2	390.50	15.71	21.7
14	0.62976	0.0	8.14	0	0.5380	5.949	61.8	4.7075	4	307	21.0	396.90	8.26	20.4
15	0.63796	0.0	8.14	0	0.5380	6.096	84.5	4.4619	4	307	21.0	380.02	10.26	18.2
16	0.62739	0.0	8.14	0	0.5380	5.834	56.5	4.4986	4	307	21.0	395.62	8.47	19.9
17	1.05393	0.0	8.14	0	0.5380	5.935	29.3	4.4986	4	307	21.0	386.85	6.58	23.1
18	0.78420	0.0	8.14	0	0.5380	5.990	81.7	4.2579	4	307	21.0	386.75	14.67	17.5
19	0.80271	0.0	8.14	0	0.5380	5.456	36.6	3.7965	4	307	21.0	288.99	11.69	20.2
20	0.72580	0.0	8.14	0	0.5380	5.727	69.5	3.7965	4	307	21.0	390.95	11.28	18.2
21	1.25179	0.0	8.14	0	0.5380	5.570	98.1	3.7979	4	307	21.0	376.57	21.02	13.6
22	0.85204	0.0	8.14	0	0.5380	5.965	89.2	4.0123	4	307	21.0	392.53	13.83	19.6
23	1.23247	0.0	8.14	0	0.5380	6.142	91.7	3.9769	4	307	21.0	396.90	18.72	15.2
24	0.98843	0.0	8.14	0	0.5380	5.813	100.0	4.0952	4	307	21.0	394.54	19.88	14.5
25	0.75026	0.0	8.14	0	0.5380	5.924	94.1	4.3996	4	307	21.0	394.33	16.30	15.6
26	0.84054	0.0	8.14	0	0.5380	5.599	85.7	4.4546	4	307	21.0	303.42	16.51	13.9
27	0.67191	0.0	8.14	0	0.5380	5.813	90.3	4.6820	4	307	21.0	376.88	14.81	16.6
28	0.95577	0.0	8.14	0	0.5380	6.047	88.8	4.4534	4	307	21.0	306.38	17.28	14.8
29	0.77299	0.0	8.14	0	0.5380	6.495	94.4	4.4547	4	307	21.0	387.94	12.80	18.4
30	1.00245	0.0	8.14	0	0.5380	6.674	87.3	4.2390	4	307	21.0	380.23	11.98	21.0

Showing 1 to 31 of 506 entries

Principle Component Analysis



Simple Example:
Convert 2D into 1D

```r
# Define target (y) and features (x)
target <- "medv" # median house value
features <- setdiff(colnames(h_boston), target)
print(features)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "b"       "lstat"
```

Principal Component Analysis in H_2O

```
# Run a Principal Components Analysis (PCA)
model_pca <- h2o.prcomp(training_frame = h_boston,
                        x = features,
                        k = 2,
                        transform = "STANDARDIZE",
                        pca_method = "GLRM",
                        use_all_factor_levels = TRUE,
                        seed = 1234)

# Extract the first two principle components
h_pca <- h2o.predict(model_pca, h_boston)

# Visualise
d_pca <- as.data.frame(h_pca)
plot(d_pca, main = "First Two Principle Components of Boston Housing Data")
```

Compress 13D into 2D

New algorithm from Stanford Research


First Two Principle Components of Boston Housing Data
PC2
0.5
0.0
-0.5
-4
-2
0
2
4
6
PC1

R Script: ***outlier_detection.R***

```
# Training a Deep Autoencoder
model <- h2o.deeplearning(x = features,
                          training_frame = h_boston,
                          autoencoder = TRUE,
                          activation = "Tanh",
                          hidden = c(100, 100, 100),
                          epochs = 100,
                          seed = 1234,
                          reproducible = TRUE)

# Calculate reconstruction errors (MSE)
recon_errors <- h2o.anomaly(model, h_boston, per_feature = FALSE)
print(recon_errors)
```

```
##   Reconstruction.MSE
## 1       0.0005537913
## 2       0.0005078867
## 3       0.0006769128
## 4       0.0005546471
## 5       0.0006467393
## 6       0.0004523183
##
## [506 rows x 1 column]
```

R Script: ***outlier_detection.R***

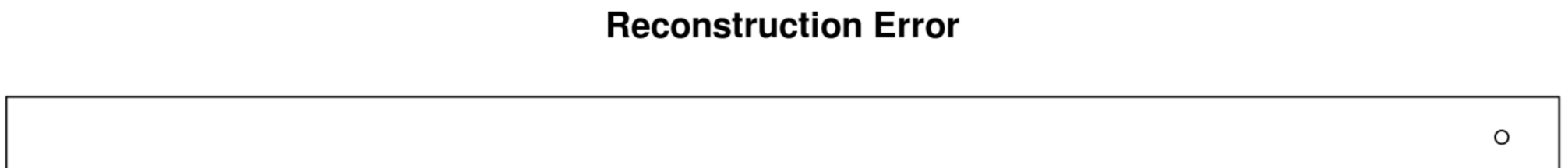
```r
# Convert H2O data frame into R data types
d_errors <- as.data.frame(recon_errors)
n_errors <- as.numeric(recon_errors)

# user-defined cut-off point
cutoff <- quantile(n_errors, probs = 0.95)

# Identify Outliers
row_outliers <- which(d_errors > cutoff) # based on plot above
length(row_outliers)
```

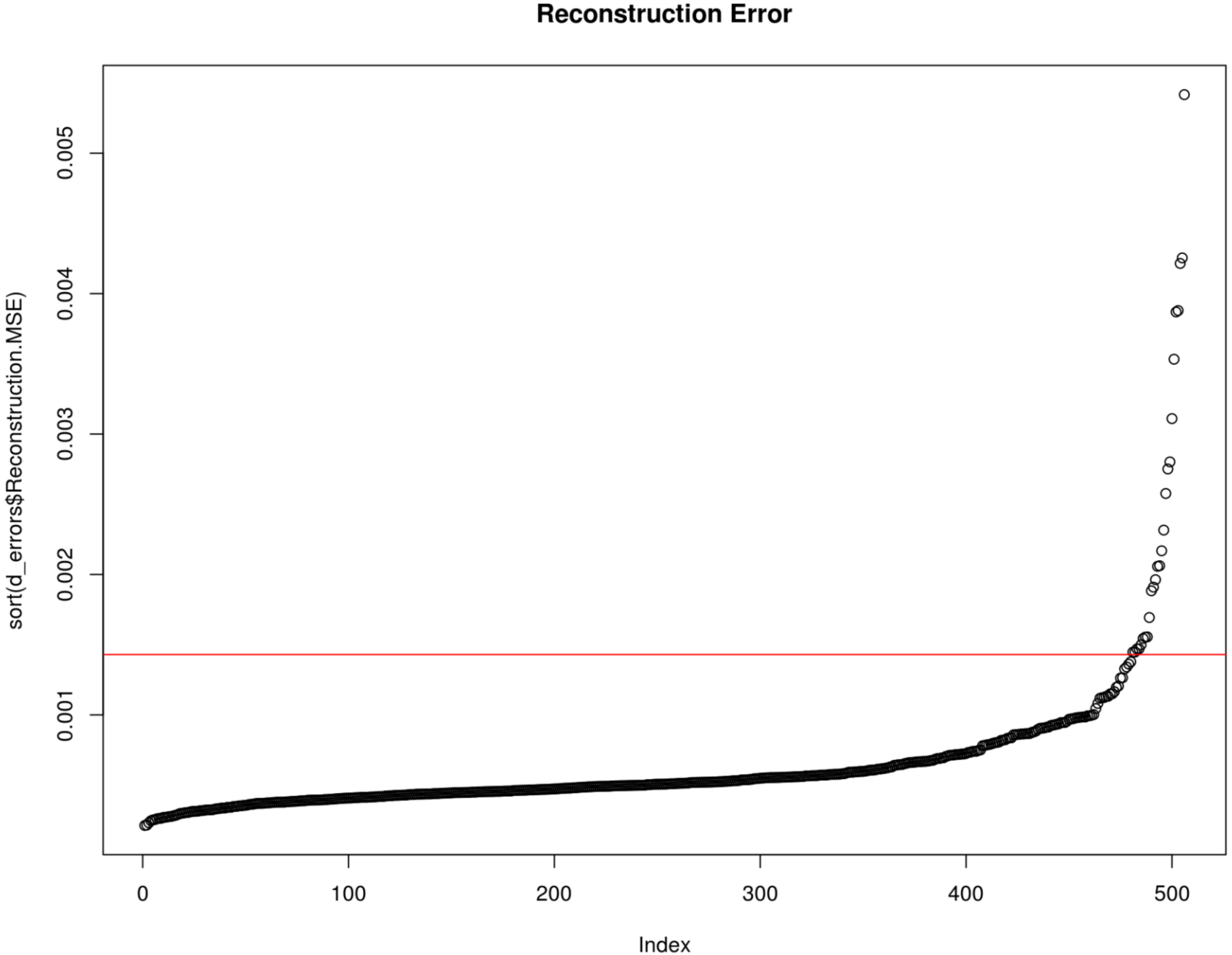
User's Choice

```
## [1] 26
```

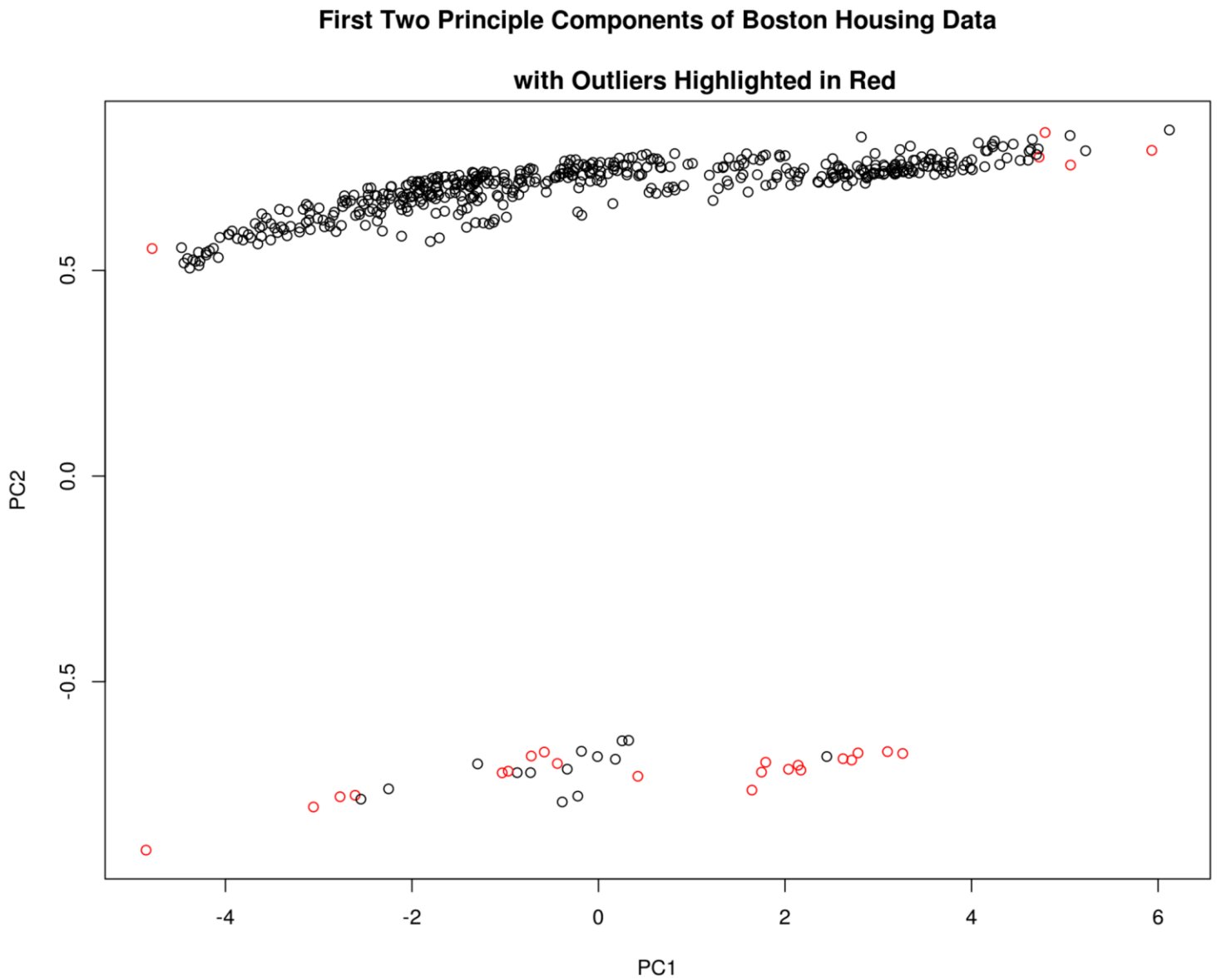
R Script: ***outlier_detection.R***

```r
# Plot Reconstruction Errors and Cutoff
plot(sort(d_errors$Reconstruction.MSE), main = "Reconstruction Error")
abline(h = cutoff, col = "red") # red line = cutoff point
```




Reconstruction Error
sort(d_errors$Reconstruction.MSE)
0.001
0.002
0.003
0.004
0.005
0
100
200
300
400
500
Index

```r
# Visualise
d_pca$outlier <- 0
d_pca[row_outliers, ]$outlier <- 1
plot(d_pca[, 1:2], col = as.factor(d_pca$outlier),
     main = "First Two Principle Components of Boston Housing Data\n
             with Outliers Highlighted in Red")
```

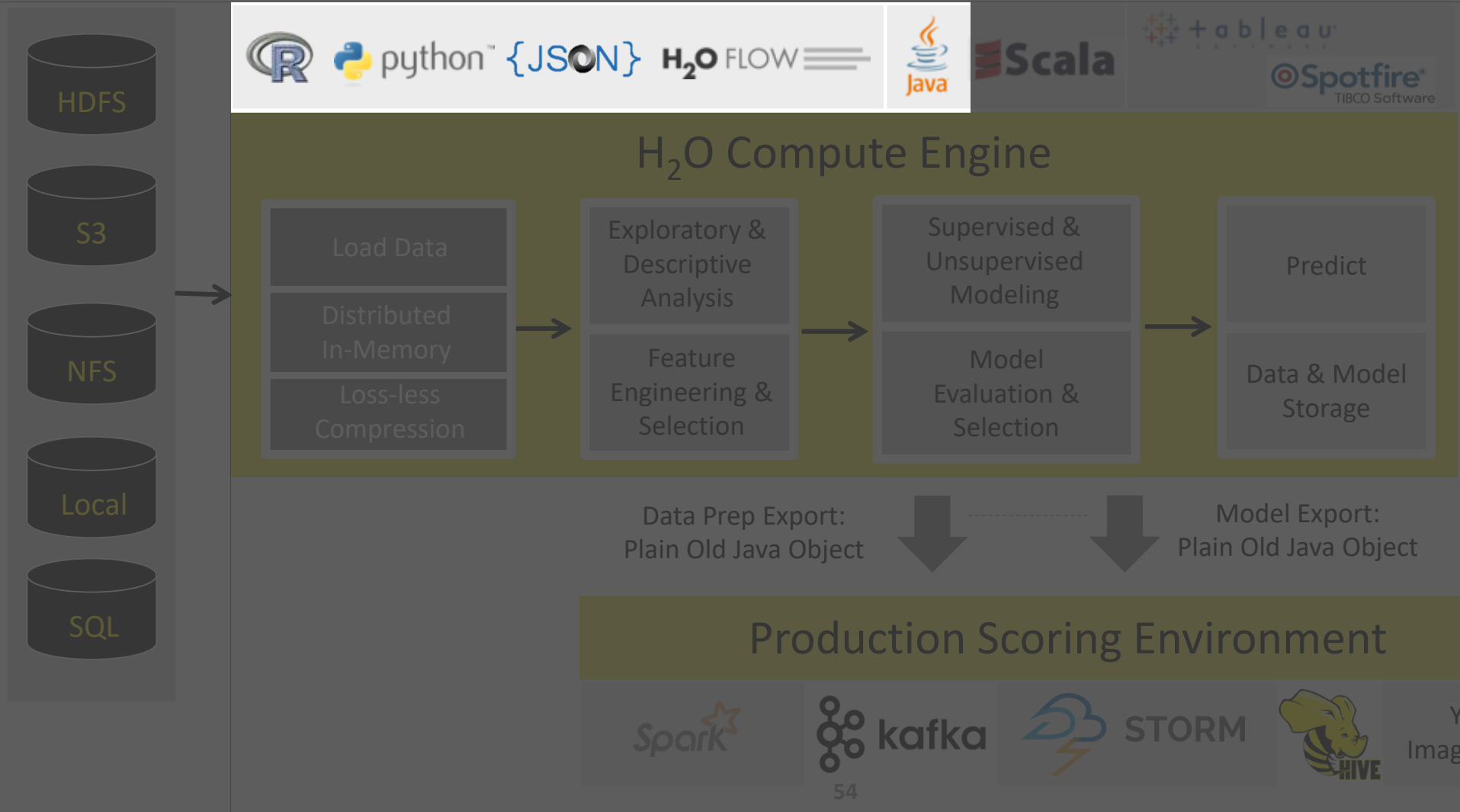


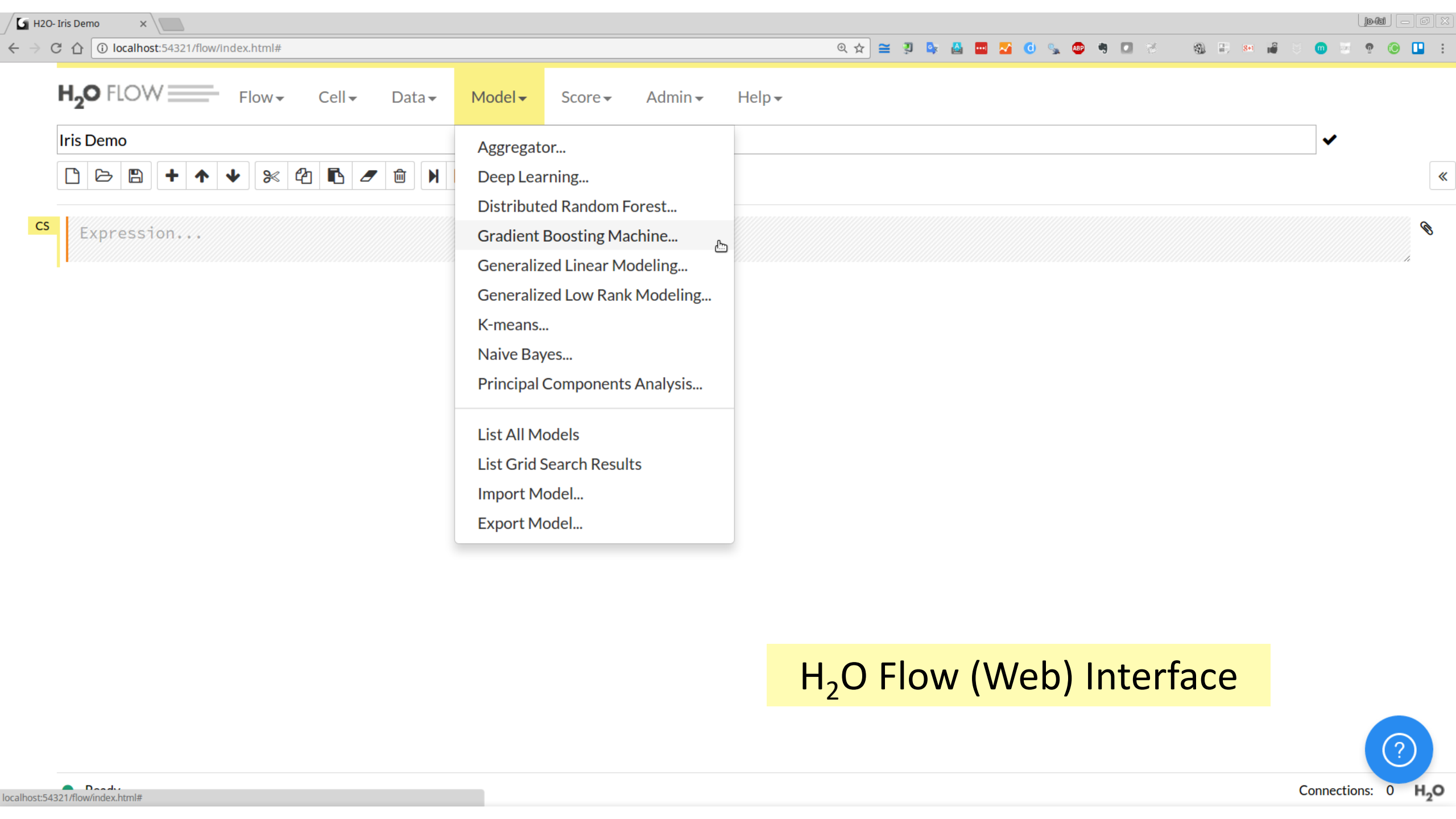
Introduction to Deep Water

H_2O Machine Learning Platform

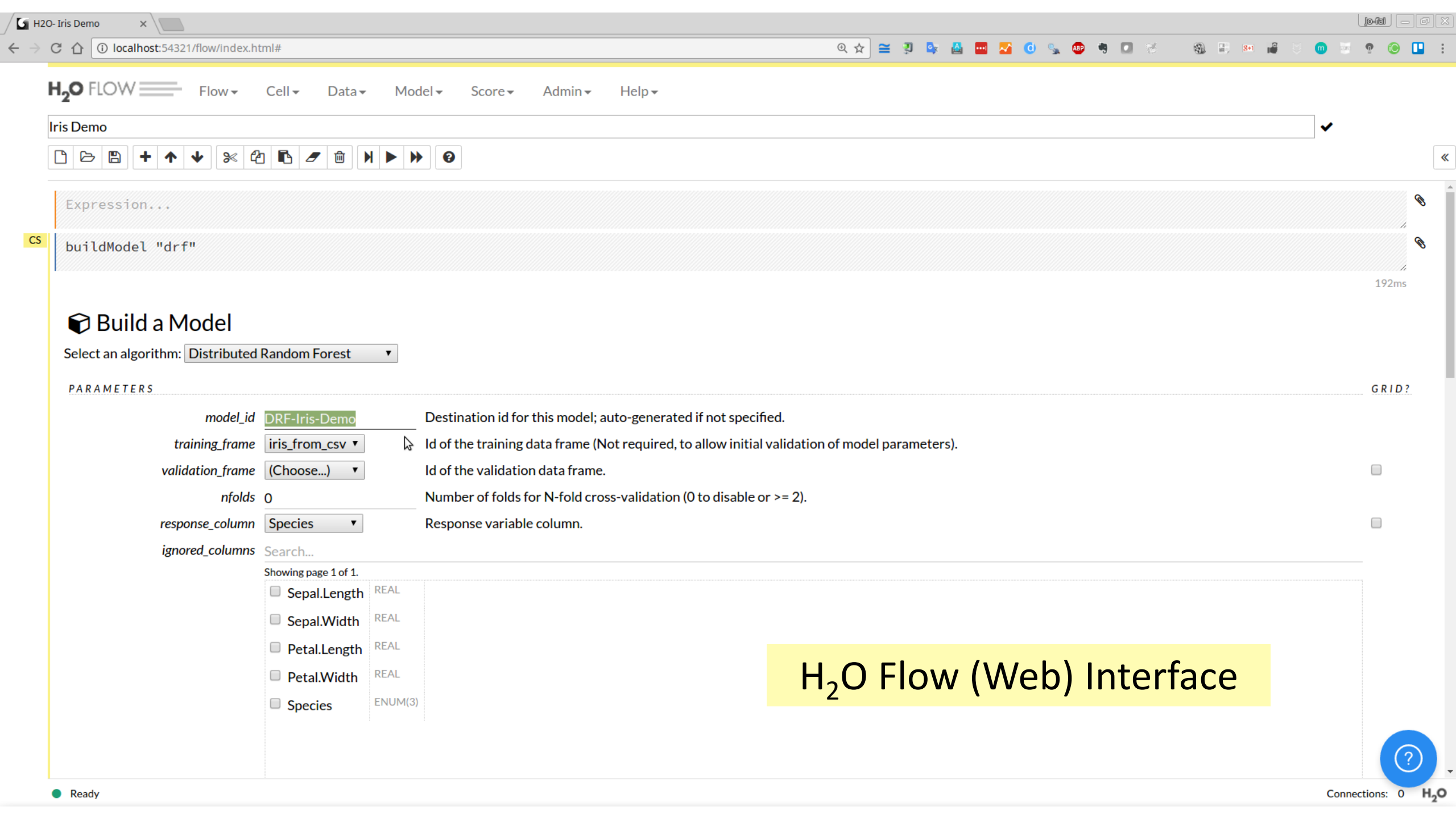
High Level Architecture

Flow (Web), R, Python API
Java for computation





H_2O Flow (Web) Interface


H2O- Iris Demo
localhost:54321/flow/index.html#
H2O FLOW
Flow
Cell
Data
Model
Score
Admin
Help
Iris Demo
Expression...
CS
buildModel "drf"
192ms
Build a Model
Select an algorithm: Distributed Random Forest
PARAMETERS
GRID?
model_id
DRF-Iris-Demo
Destination id for this model; auto-generated if not specified.
training_frame
iris_from_csv
Id of the training data frame (Not required, to allow initial validation of model parameters).
validation_frame
(Choose...)
Id of the validation data frame.
nfolds
0
Number of folds for N-fold cross-validation (0 to disable or >= 2).
response_column
Species
Response variable column.
ignored_columns
Search...
Showing page 1 of 1.
Sepal.Length REAL
Sepal.Width REAL
Petal.Length REAL
Petal.Width REAL
Species ENUM(3)
H2O Flow (Web) Interface
Ready
Connections: 0

H_2O + R

h2o_iris_demo.R

Source on Save | Run | Source

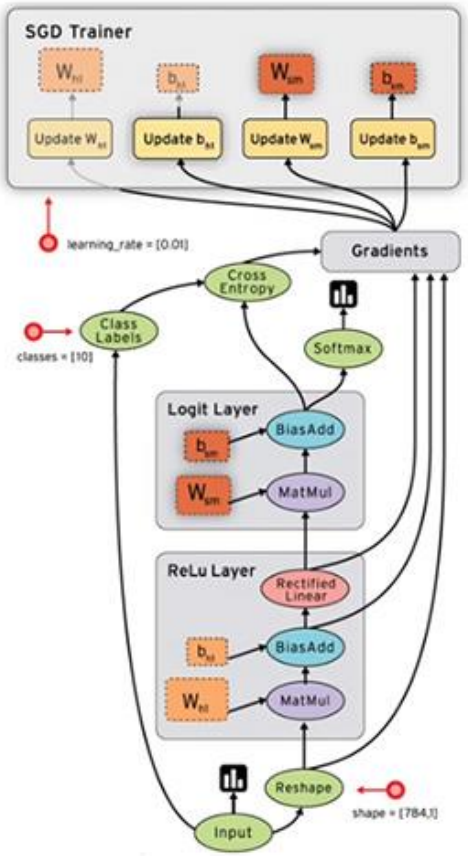
```r
# ------------------------------------------------------------------------------
# Build a simple classification model using iris dataset
# ------------------------------------------------------------------------------

# Start and connect to a local H2O cluster
library(h2o)
h2o.init(nthreads = -1)

# Import data from a R data frame
data(iris)
d_iris <- as.h2o(iris)

# Define Targets and Features
target <- "Species"
features <- setdiff(colnames(d_iris), c("Species"))

# ------------------------------------------------------------------------------
# Train a H2O Model
# ------------------------------------------------------------------------------

# Train three basic H2O models
model_drf <- h2o.randomForest(x = features,
                              y = target,
                              model_id = "iris_random_forest",
                              training_frame = d_iris)

model_gbm <- h2o.gbm(x = features,
                     y = target,
                     model_id = "iris_gbm",
                     training_frame = d_iris)

model_dnn <- h2o.deeplearning(x = features,
                              y = target,
                              model_id = "iris_deep_learning",
                              training_frame = d_iris)
```

Deep Learning Tools

TensorFlow, mxnet, Caffe and H_2O Deep Learning

TensorFlow

- Open source machine learning framework by Google
- Python / C++ API
- TensorBoard
  - Data Flow Graph Visualization
- Multi CPU / GPU
  - v0.8+ distributed machines support
- Multi devices support
  - desktop, server and Android devices
- Image, audio and NLP applications
- **HUGE** Community
- Support for Spark, Windows …



https://github.com/tensorflow/tensorflow

TensorFlow Wrappers

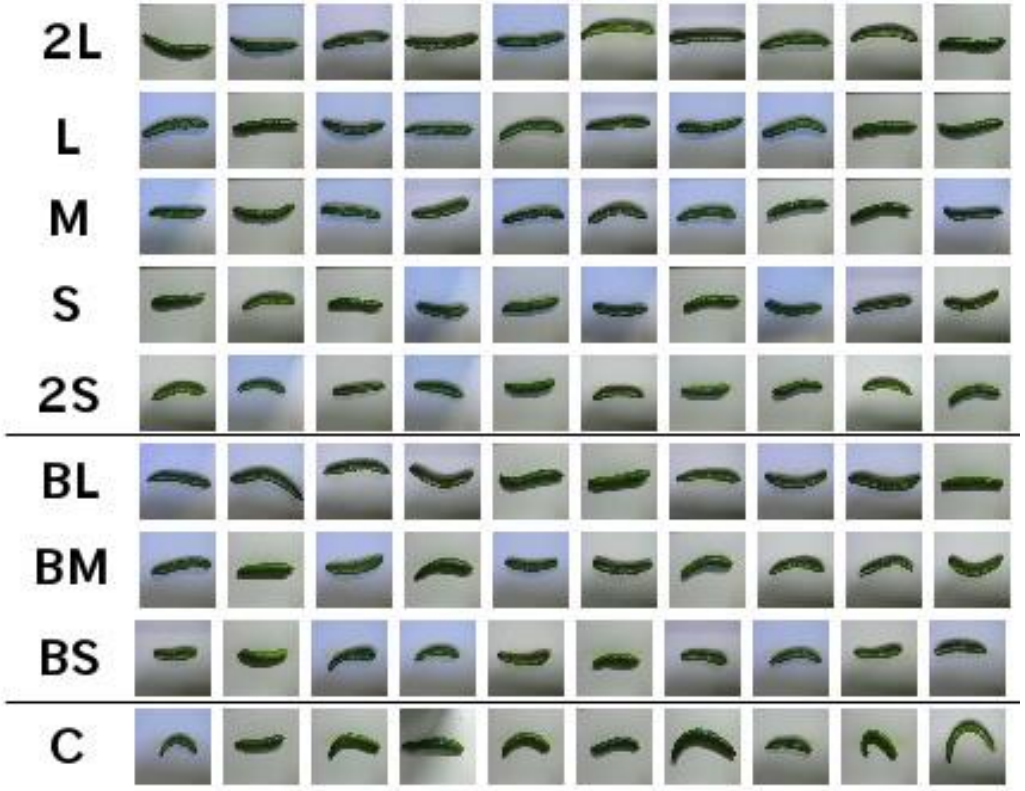
- TFLearn – Simplified interface
- keras – TensorFlow + Theano
- tensorflow.rb – Ruby wrapper
- TensorFlow.jl – Julia wrapper
- TensorFlow for R – R wrapper

- … and many more!
- See: github.com/jtoy/awesome-tensorflow

Sorting Cucumbers

- Problem
  - Sorting cucumbers is a laborious process.
  - In a Japanese farm, the farmer's wife can spend up to eight hours a day sorting cucumbers during peak harvesting period.
- Solution
  - Farmer's son (Makoto Koike) used TensorFlow, Arduino and Raspberry Pi to create an automatic cucumber sorting system.

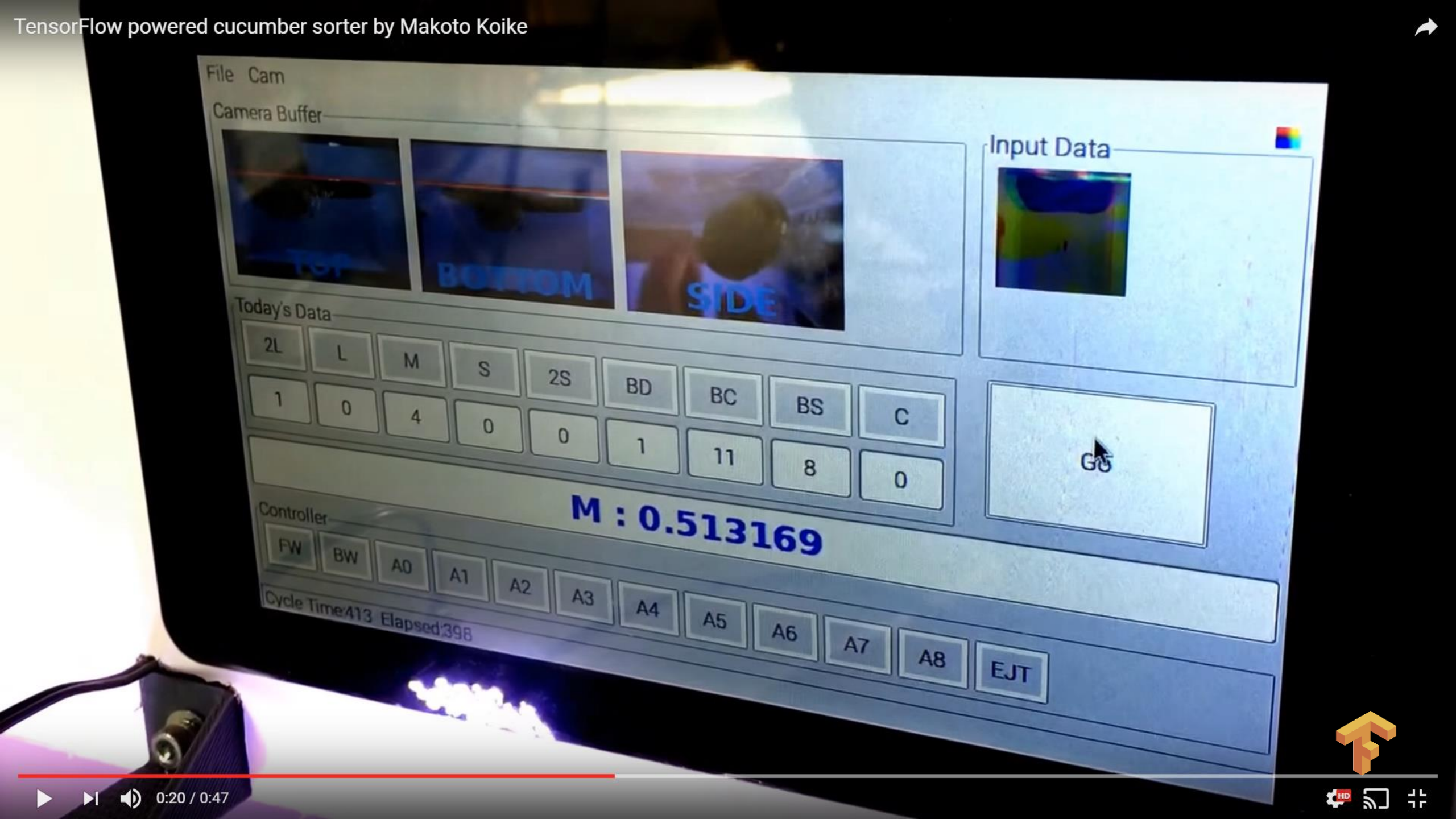
Sorting Cucumbers



- Classification Problem
  - Input: cucumber photos (side, top, bottom)
  - Output: one of nine classes

- Google's Blog Post [Link]
- YouTube Video [Link]


TensorFlow powered cucumber sorter by Makoto Koike
0:02 / 0:47


TensorFlow powered cucumber sorter by Makoto Koike
File Cam
Camera Buffer
TOP
BOTTOM
SIDE
Input Data
Today's Data
2L L M S 2S BD BC BS C
1 0 4 0 0 1 11 8 0
GO
M : 0.513169
Controller
FW BW A0 A1 A2 A3 A4 A5 A6 A7 A8 EJT
Cycle Time:413 Elapsed:398
0:20 / 0:47


TensorFlow powered cucumber sorter by Makoto Koike
宅急便
Bou
0:10 / 0:47

TensorKart – Self-driving Mario Kart


Kevin Hughes, the developer behind TensorKart

"The idea of exploring AI techniques in video games was not new, but what motivated me to do this project was to showcase complete pipeline of a machine learning system. I wanted to pick a popular game because I thought it might interest more people and expose them to how machine learning works," reveals Hughes.

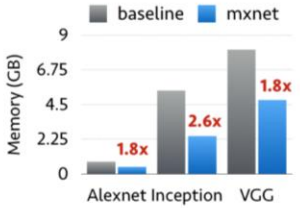
https://github.com/kevinhughes27/TensorKart
https://opensourceforu.com/2017/01/tensorflow-brings-self-driving-to-mario-kart/

dmlc mxnet for Deep Learning

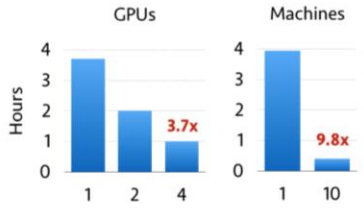
build passing | docs latest | license Apache 2.0

Portable

Efficient



Scalable



MXNet is a deep learning framework designed for both *efficiency* and *flexibility*. It allows you to ***mix*** the flavours of symbolic programming and imperative programming to ***maximize*** efficiency and productivity. In its core, a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on the fly. A graph optimization layer on top of that makes symbolic execution fast and memory efficient. The library is portable and lightweight, and it scales to multiple GPUs and multiple machines.

MXNet is also more than a deep learning project. It is also a collection of blue prints and guidelines for building deep learning system, and interesting insights of DL systems for hackers.

MXNet now chosen by Amazon as Deep Learning Framework

By Geneva Clark | 2016-11-24 ♡19 💬0

Share this magazine

Amazon has announced that it has chosen MXNet as its deep learning framework of choice for its web services(AWS). Amazon extensively uses machine learning in areas like fraud detection, abusive review detection, and book classification. Amazon also uses it in application areas such as text and speech recognition, autonomous drones etc...

https://github.com/dmlc/mxnet
https://www.zeolearn.com/magazine/amazon-to-use-mxnet-as-deep-learning-framework

H2O.ai

Caffe

- **C**onvolution **A**rchitecture **F**or **F**eature **E**xtraction (CAFFE)
- Pure C++ / CUDA architecture for deep learning
- Command line, Python and MATLAB interface
- Model Zoo
  - Open collection of models

https://docs.google.com/presentation/d/1UeKXVgRvvxg9OUdh_UiC5G71UMscNPlvArsWER41PsU/

H2O.ai

H_2O Deep Learning

Supervised Learning

Statistical Analysis	• **Generalized Linear Models**: Binomial, Gaussian, Gamma, Poisson and Tweedie • **Naïve Bayes**
Ensembles	• **Distributed Random Forest**: Classification or regression models • **Gradient Boosting Machine**: Produces an ensemble of decision trees with increasing refined approximations
Deep Neural Networks	• **Deep learning**: Create multi-layer feed forward neural networks starting with an input layer followed by multiple layers of nonlinear transformations

Unsupervised Learning

Clustering	• **K-means**: Partitions observations into k clusters/groups of the same spatial size. Automatically detect optimal k
Dimensionality Reduction	• **Principal Component Analysis**: Linearly transforms correlated variables to independent components • **Generalized Low Rank Models:** extend the idea of PCA to handle arbitrary data consisting of numerical, Boolean, categorical, and missing data
Anomaly Detection	• **Autoencoders**: Find outliers using a nonlinear dimensionality reduction using deep learning

TensorFlow , MXNet, Caffe and H_2O democratize the power of deep learning.

H_2O platform democratizes artificial intelligence & big data science.

There are other open source deep learning libraries like Theano and Torch too.

Let’s have a party, this will be fun!

H_2O.ai


H2O.ai
Products
Industries
Customers
Community
Documentation
Download
Deep Water
H2O.ai
Caffe
mxnet
TensorFlow

Deep Water

Next-Gen Distributed Deep Learning with H_2O

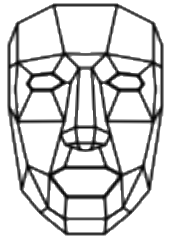
One Interface - GPU Enabled - Significant Performance Gains

Inherits All H_2O Properties in Scalability, Ease of Use and Deployment

TensorFlow Caffe H_2O.ai mxnet

Deep Water

H_2O integrates with existing GPU backends for significant performance gains



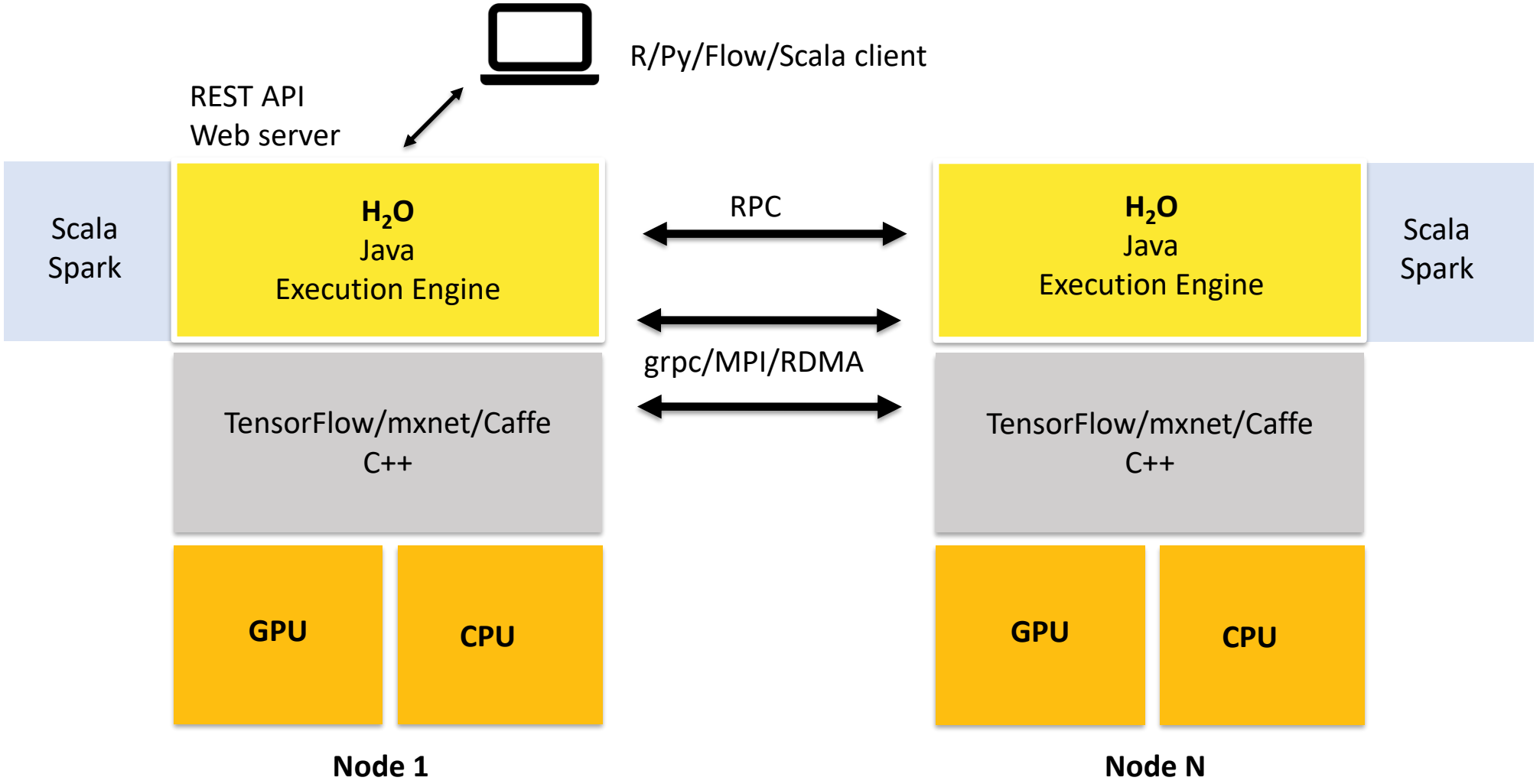
Convolutional Neural Networks enabling **Image, video, speech recognition**

Recurrent Neural Networks enabling **natural language processing, sequences, time series,** and more

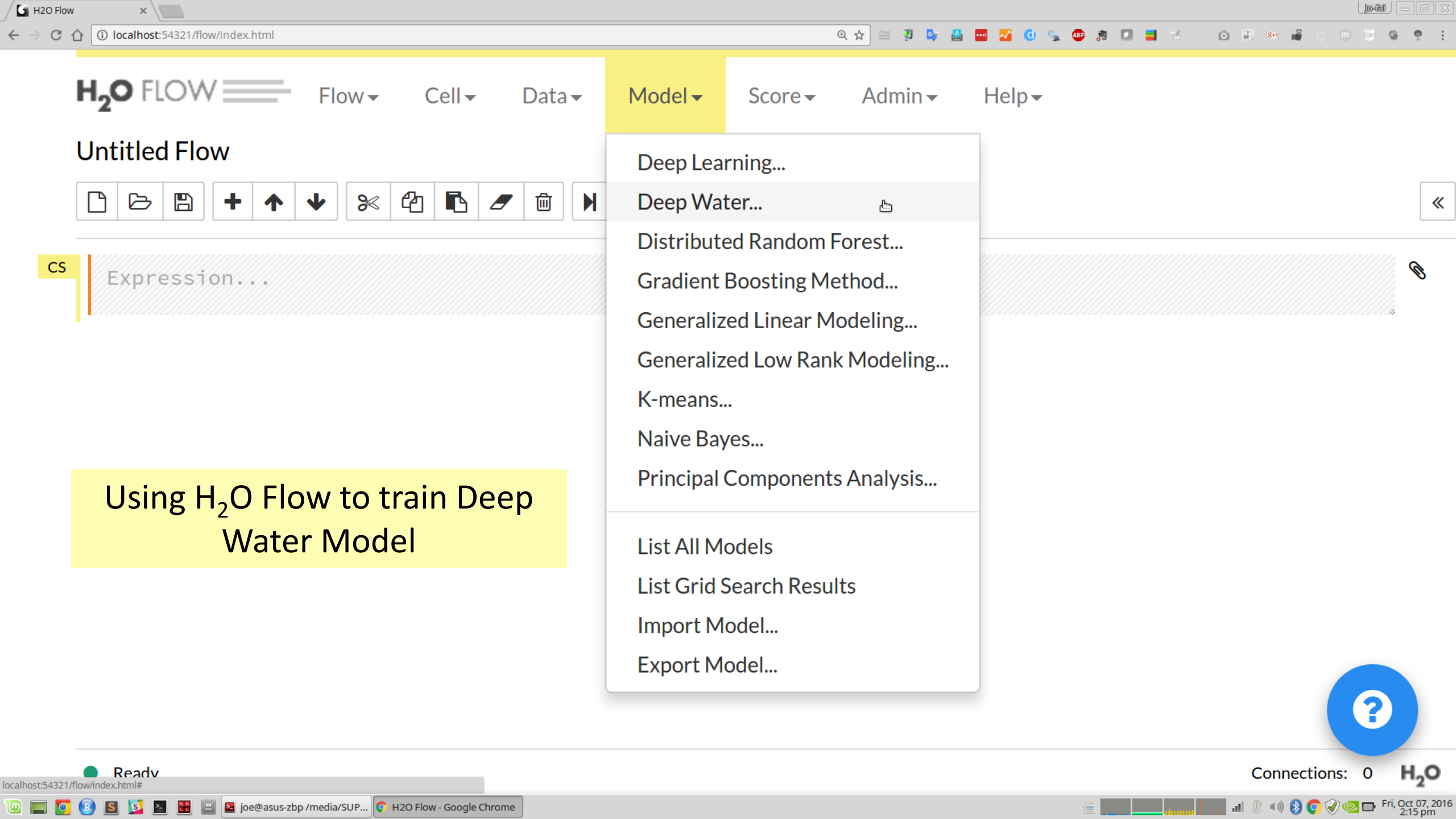
Hybrid Neural Network Architectures enabling **speech to text translation, image captioning, scene parsing** and more

H_2O.ai

Deep Water Architecture

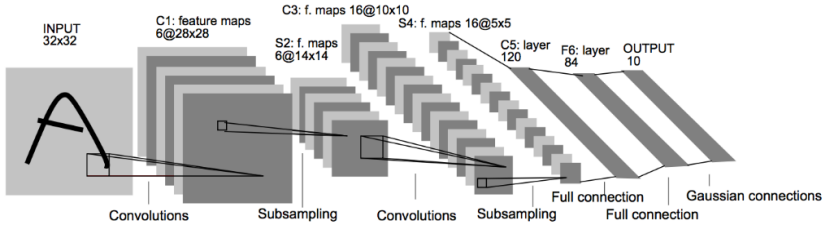


H2O.ai

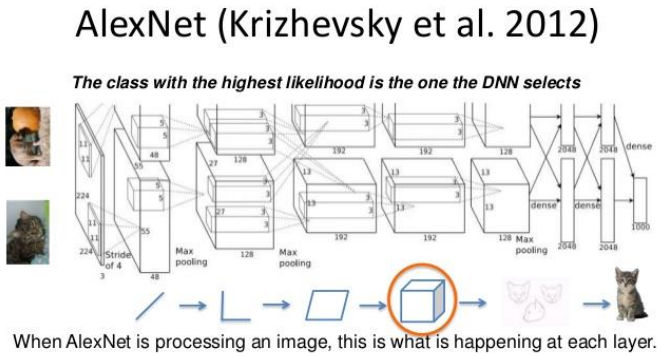

H2O Flow
localhost:54321/flow/index.html
H2O FLOW
Flow
Cell
Data
Model
Score
Admin
Help
Untitled Flow
CS
Expression...
Deep Learning...
Deep Water...
Distributed Random Forest...
Gradient Boosting Method...
Generalized Linear Modeling...
Generalized Low Rank Modeling...
K-means...
Naive Bayes...
Principal Components Analysis...
List All Models
List Grid Search Results
Import Model...
Export Model...
Using H2O Flow to train Deep Water Model
Ready
Connections: 0
H2O
localhost:54321/flow/index.html#
joe@asus-zbp /media/SUP...
H2O Flow - Google Chrome
Fri, Oct 07, 2016
2:15 pm

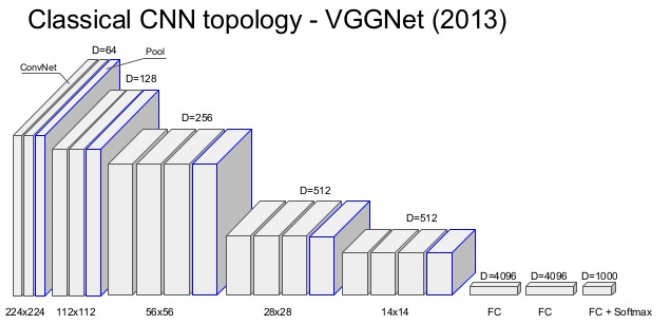
Available Networks in Deep Water

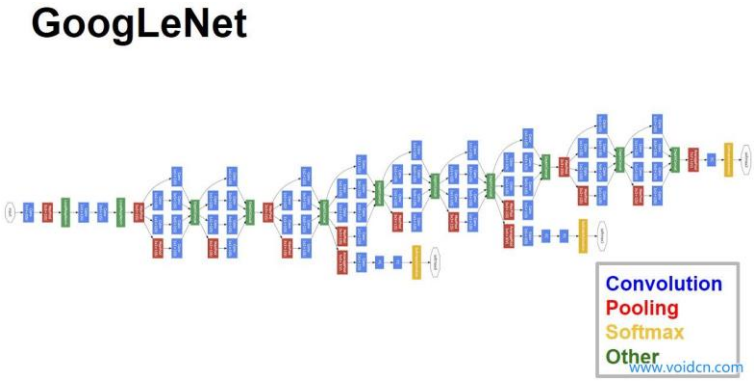
- LeNet
- AlexNet
- VGGNet
- Inception (GoogLeNet)
- ResNet (Deep Residual Learning)
- Build Your Own

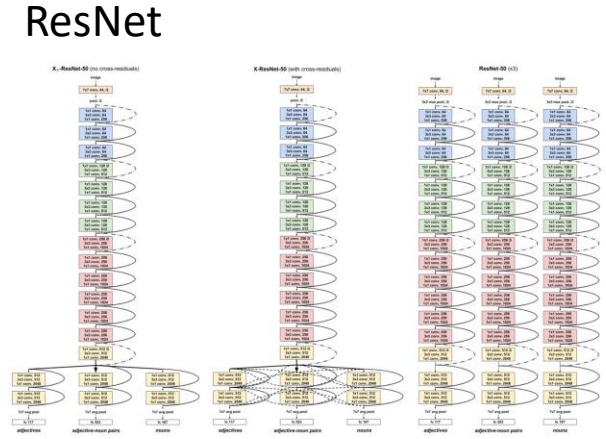


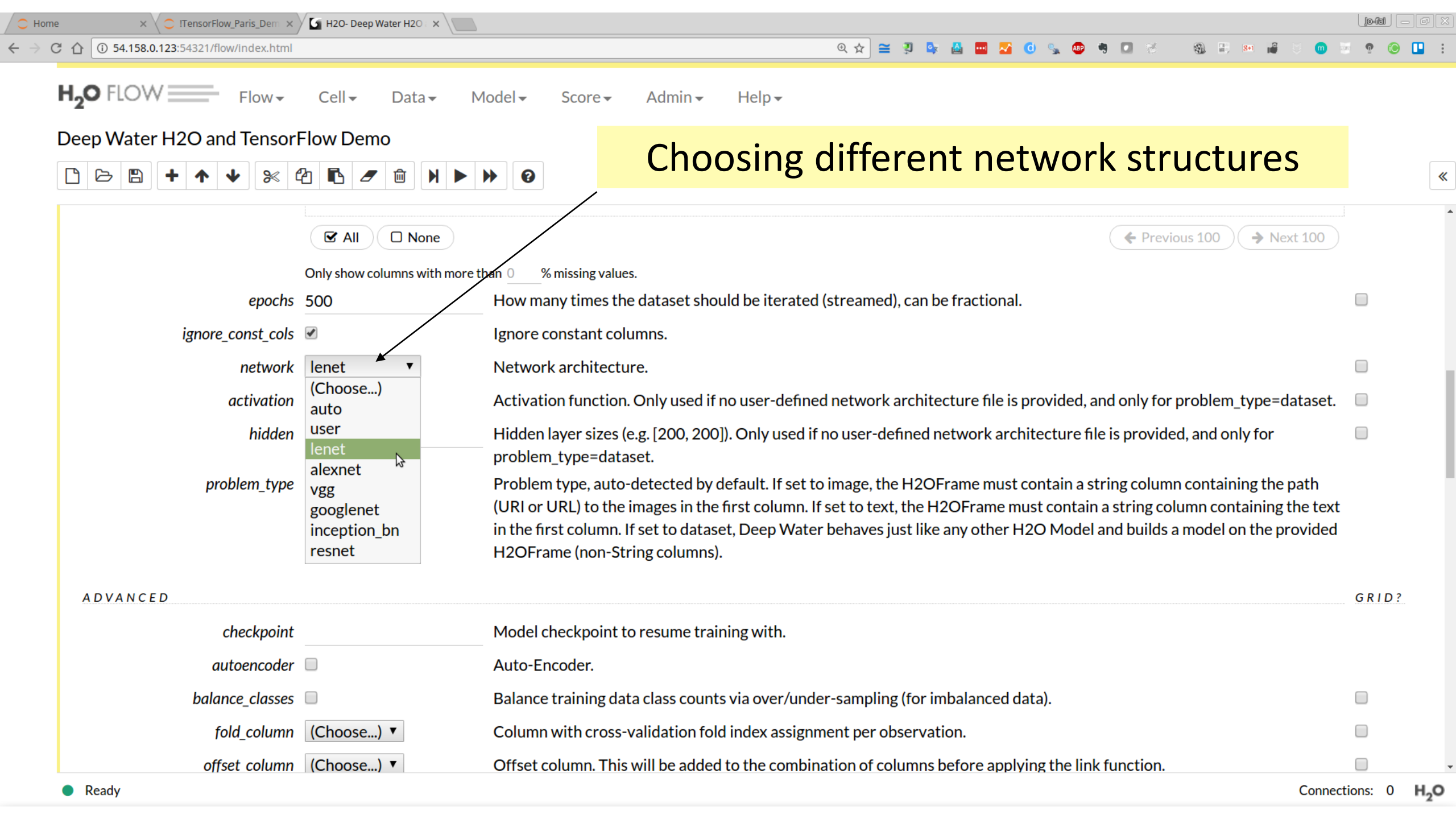
CNN called LeNet by Yann LeCun (1998)

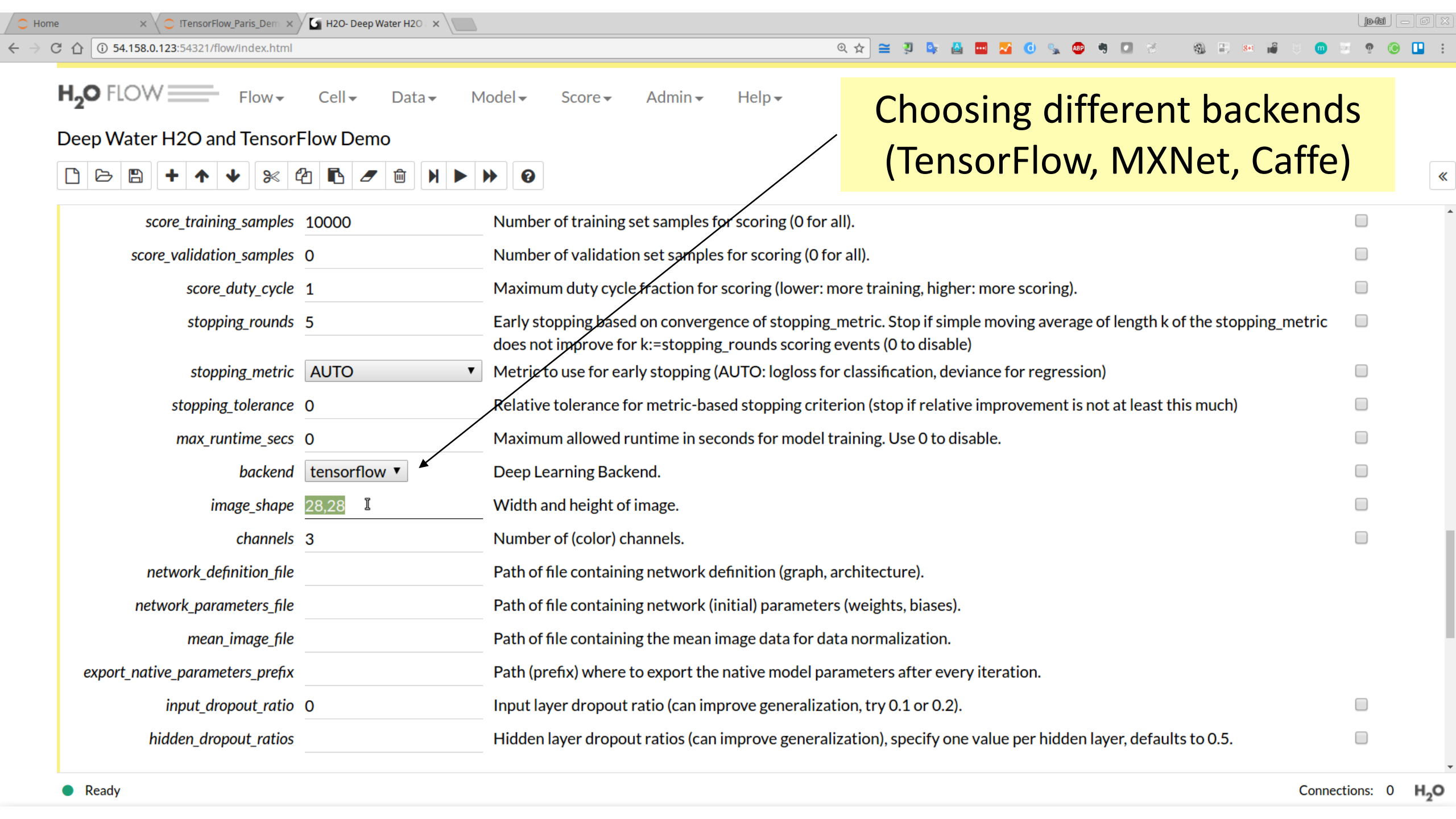









Home
ITensorFlow_Paris_Dem
H2O- Deep Water H2O
54.158.0.123:54321/flow/index.html
H2O FLOW
Flow
Cell
Data
Model
Score
Admin
Help
Deep Water H2O and TensorFlow Demo
Choosing different network structures
All
None
Previous 100
Next 100
Only show columns with more than 0 % missing values.
epochs
500
How many times the dataset should be iterated (streamed), can be fractional.
ignore_const_cols
Ignore constant columns.
network
lenet
Network architecture.
activation
Activation function. Only used if no user-defined network architecture file is provided, and only for problem_type=dataset.
hidden
Hidden layer sizes (e.g. [200, 200]). Only used if no user-defined network architecture file is provided, and only for problem_type=dataset.
problem_type
Problem type, auto-detected by default. If set to image, the H2OFrame must contain a string column containing the path (URI or URL) to the images in the first column. If set to text, the H2OFrame must contain a string column containing the text in the first column. If set to dataset, Deep Water behaves just like any other H2O Model and builds a model on the provided H2OFrame (non-String columns).
(Choose...)
auto
user
lenet
alexnet
vgg
googlenet
inception_bn
resnet
ADVANCED
GRID?
checkpoint
Model checkpoint to resume training with.
autoencoder
Auto-Encoder.
balance_classes
Balance training data class counts via over/under-sampling (for imbalanced data).
fold_column
(Choose...)
Column with cross-validation fold index assignment per observation.
offset_column
(Choose...)
Offset column. This will be added to the combination of columns before applying the link function.
Ready
Connections: 0


Home
ITensorFlow_Paris_Dem
H2O- Deep Water H2O
54.158.0.123:54321/flow/index.html
H2O FLOW
Flow
Cell
Data
Model
Score
Admin
Help
Deep Water H2O and TensorFlow Demo
Choosing different backends
(TensorFlow, MXNet, Caffe)
score_training_samples 10000 Number of training set samples for scoring (0 for all).
score_validation_samples 0 Number of validation set samples for scoring (0 for all).
score_duty_cycle 1 Maximum duty cycle fraction for scoring (lower: more training, higher: more scoring).
stopping_rounds 5 Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k:=stopping_rounds scoring events (0 to disable)
stopping_metric AUTO Metric to use for early stopping (AUTO: logloss for classification, deviance for regression)
stopping_tolerance 0 Relative tolerance for metric-based stopping criterion (stop if relative improvement is not at least this much)
max_runtime_secs 0 Maximum allowed runtime in seconds for model training. Use 0 to disable.
backend tensorflow Deep Learning Backend.
image_shape 28,28 Width and height of image.
channels 3 Number of (color) channels.
network_definition_file Path of file containing network definition (graph, architecture).
network_parameters_file Path of file containing network (initial) parameters (weights, biases).
mean_image_file Path of file containing the mean image data for data normalization.
export_native_parameters_prefix Path (prefix) where to export the native model parameters after every iteration.
input_dropout_ratio 0 Input layer dropout ratio (can improve generalization, try 0.1 or 0.2).
hidden_dropout_ratios Hidden layer dropout ratios (can improve generalization), specify one value per hidden layer, defaults to 0.5.
Ready
Connections: 0
H2O

Unified Interface for TF, MXNet and Caffe

Choosing different network structures

```
model = H2ODeepWaterEstimator(epochs      = 500,
                              network     = "lenet",
                              image_shape = [28,28],  ## provide image size
                              channels    = 3,
                              backend     = "tensorflow",
                              model_id    = "deepwater_tf_simple")

model.train(x = [0], # file path e.g. xxx/xxx/xxx.jpg
            y = 1, # label cat/dog/mouse
            training_frame = frame)

model.show()
```

```
deepwater Model Build progress: |████████████████████████████████████████| 100%
Model Details
=============
H2ODeepWaterEstimator :  Deep Water
Model Key:  deepwater_tf_simple
```

Change backend to “mxnet”, “caffe” or “auto”

Easy Stacking with other H_2O Models

Model Stacking

Now we have three different models, we are ready to carry out model stacking.

In [47]:
```
# Create a list to include all the models for stacking
models <- list(model_dw, model_gbm, model_drf)
```

In [48]:
```
# Define a metalearner (one of the H2O supervised machine learning algorithms)
metalearner <- "h2o.glm.wrapper"
```

In [49]:
```
# Use h2o.stack() to carry out metalearning
stack <- h2o.stack(models = models,
                   response_frame = h_train$medv,
                   metalearner = metalearner)
```

```
[1] "Metalearning"
```

In [50]:
```
# Finally, we evaluate the predictive performance on the ensemble as well as individual models.
h2o.ensemble_performance(stack, newdata = h_test)
```

```
Base learner performance, sorted by specified metric:
        learner      MSE
1 h2o_deepwater 8.377644
2       h2o_gbm 8.106541
3       h2o_drf 7.443517


H2O Ensemble Performance on <newdata>:
----------------
Family: gaussian

Ensemble performance (MSE): 5.80436983051916
```

Ensemble of Deep Water, Gradient Boosting Machine & Random Forest models

H2O, Sparkling Water, Steam, & Deep Water Documentation

Getting Started | Data Science Algorithms | Languages | Tutorials, Examples, & Presentations | For Developers | For the Enterprise

docs.h2o.ai

Getting Started

H2O

What is H2O?
H2O User Guide
H2O Book (O'Reilly)
Recent Changes
Open Source License (Apache V2)

Quick Start Video - Flow Web UI
Quick Start Video - R
Quick Start Video - Python

Download H2O

Sparkling Water

What is Sparkling Water?
Sparkling Water Booklet
PySparkling Readme 2.0 | 1.6
RSparkling Readme
Open Source License (Apache V2)

Quick Start Video - Scala
Quick Start Video - Python

Download Sparkling Water

Steam

What is Steam?
Steam User Guide
Recent Changes
Open Source License (AGPL)

Download Steam

Deep Water (preview)

Deep Water Readme
Deep Water AMI Guide
Open Source License (Apache V2)

Launch Deep Water AMI
(choose g2.2xlarge)

Q & A

FAQ
Community Forum
h2ostream Google Group
Issue Tracking (JIRA)
Gitter
Stack Overflow
Cross Validated

For Supported Enterprise Customers

Enterprise Support Web | Email

Branch: master ▾ h2o-3 / examples / deeplearning / notebooks / Create new file | Upload files | Find file | History

mstensmo changing the name of deeplearning_credit_card_default_risk_prediction... ... Latest commit 5568350 11 days ago

..		
images	Add cat/dog/mouse lenet example.	3 months ago
README.md	Update README.md	2 months ago
deeplearning_anomaly_detection.ip...	Update notebooks, introduce local paths to ~/h2o-3/	3 months ago
deeplearning_benchmark_mnist.ipynb	Update lenet test to remove all. Update MNIST benchmark with comments.	3 months ago
deeplearning_cat_dog_mouse_incep...	Add credit card default risk model, update other notebooks.	
deeplearning_cat_dog_mouse_lenet....	Add credit card default risk model, update other notebooks.	
deeplearning_cat_dog_mouse_lenet...	Add back model.plot() and scoring history.	
deeplearning_cifar10_vgg.ipynb	Rename notebooks.	
deeplearning_credit_card_default_ri...	changing the name of deeplearning_credit_card_default_risk_prediction...	
deeplearning_ensemble_boston_ho...	Ensemble demo using GBM, DRF and Deep Water (#676)	17 days ago
deeplearning_grid_iris.ipynb	Add two new notebooks: Lenet for R and iris grid for python	3 months ago
deeplearning_grid_iris_R.ipynb	Update R py notebook.	3 months ago
deeplearning_image_reconstruction...	Update notebooks, introduce local paths to ~/h2o-3/	3 months ago
deeplearning_mnist_convnet.ipynb	Update notebooks, introduce local paths to ~/h2o-3/	3 months ago
deeplearning_mnist_introduction.ip...	Add missing file.	3 months ago
deeplearning_tensorflow_cat_dog_...	Add tensorflow example (#529)	2 months ago
deeplearning_tensorflow_mnist.ipynb	Added MNIST example for TensorFlow	a month ago

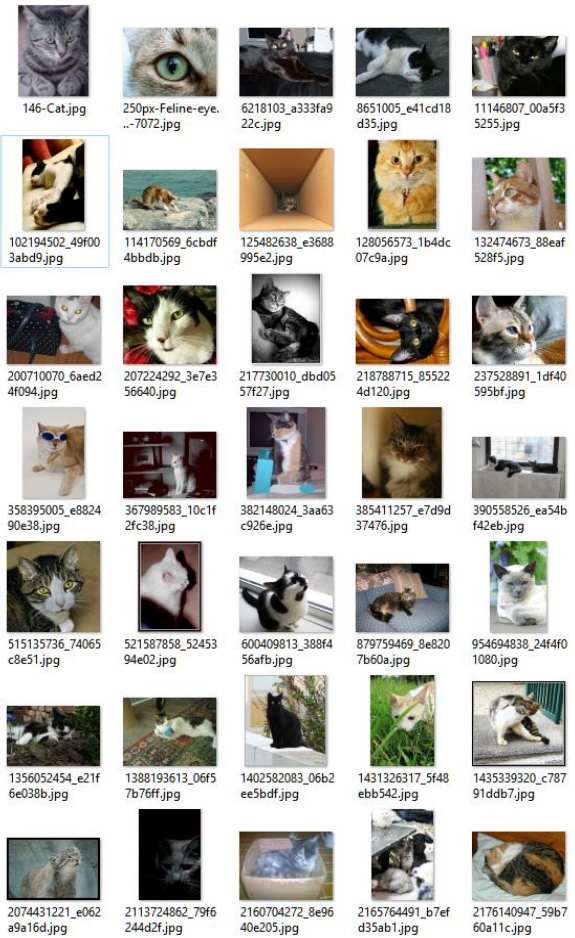
Deep Water Example notebooks

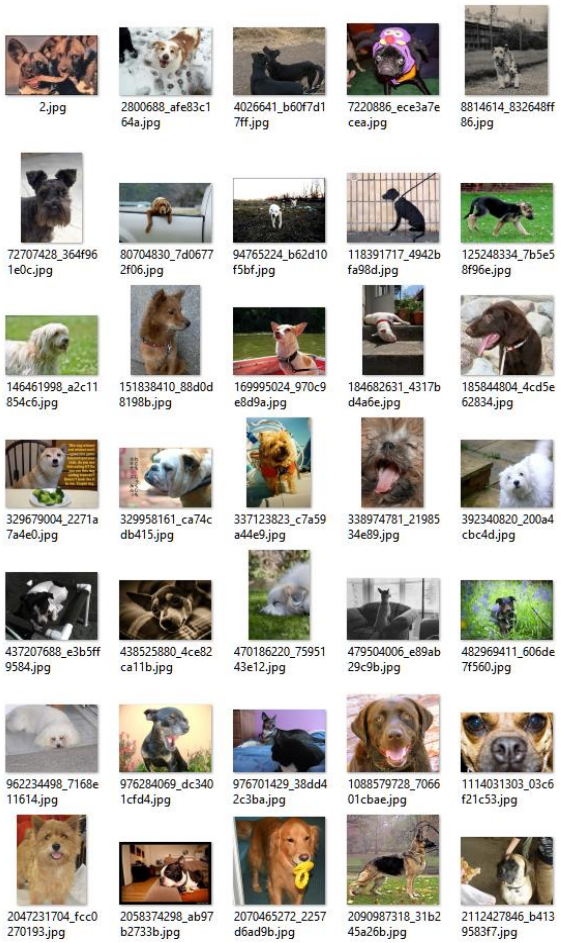
https://github.com/h2oai/h2o-3/tree/master/examples/deeplearning/notebooks

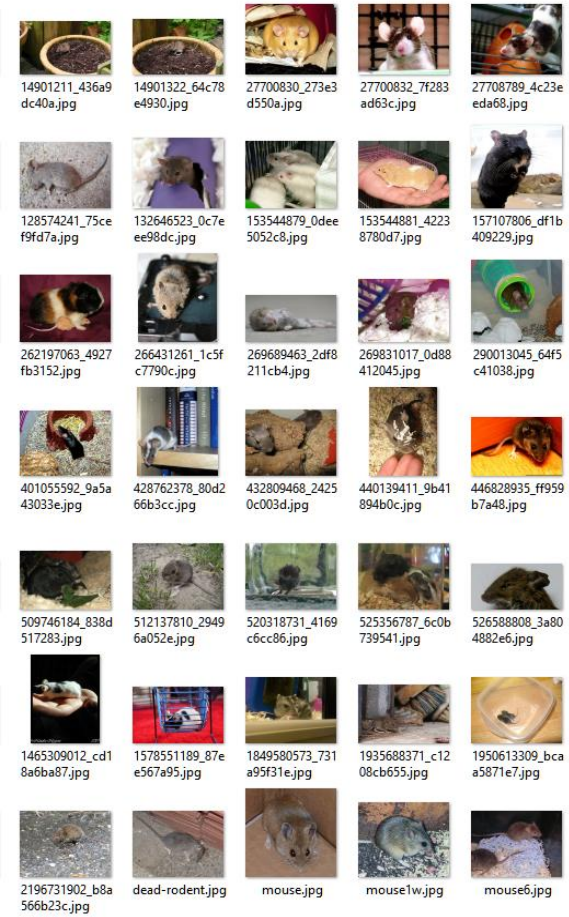
H2O.ai

Deep Water Cat/Dog/Mouse Demo

Data – Cat/Dog/Mouse Images







H2O.ai

Data – CSV

	A	B
1	bigdata/laptop/deepwater/imagenet/cat/102194502_49f003abd9.jpg	cat
2	bigdata/laptop/deepwater/imagenet/cat/11146807_00a5f35255.jpg	cat
3	bigdata/laptop/deepwater/imagenet/cat/1140846215_70e326f868.jpg	cat
4	bigdata/laptop/deepwater/imagenet/cat/114170569_6cbdf4bbdb.jpg	cat
5	bigdata/laptop/deepwater/imagenet/cat/1217664848_de4c7fc296.jpg	cat
6	bigdata/laptop/deepwater/imagenet/cat/1241603780_5e8c8f1ced.jpg	cat
7	bigdata/laptop/deepwater/imagenet/cat/1241612072_27ececbdef.jpg	cat
8	bigdata/laptop/deepwater/imagenet/cat/1241613138_ef1d82973f.jpg	cat
9	bigdata/laptop/deepwater/imagenet/cat/1244562192_35becd66bd.jpg	cat
10	bigdata/laptop/deepwater/imagenet/cat/125482638_e3688995e2.jpg	cat
11	bigdata/laptop/deepwater/imagenet/cat/128056573_1b4dc07c9a.jpg	cat
12	bigdata/laptop/deepwater/imagenet/cat/12945197_75e607e355.jpg	cat
13	bigdata/laptop/deepwater/imagenet/cat/132474673_88eaf528f5.jpg	cat
14	bigdata/laptop/deepwater/imagenet/cat/1350530984_ecf3039cf0.jpg	cat
15	bigdata/laptop/deepwater/imagenet/cat/1351606235_c9fbebf634.jpg	cat
16	bigdata/laptop/deepwater/imagenet/cat/1356052454_e21f6e038b.jpg	cat
17	bigdata/laptop/deepwater/imagenet/cat/1388193613_06f57b76ff.jpg	cat

Using Tensorflow with H2O

This notebook shows how to use the tensorflow backend to tackle a simple image classification problem.

We start by connecting to our h2o cluster:

In [1]:
```
import h2o
h2o.init(port=54321, nthreads=-1)
```

```
Checking whether there is an H2O instance running at http://localhost:54321. connected.
```

H2O cluster uptime:	54 mins 37 secs
H2O cluster version:	3.11.0.99999
H2O cluster version age:	6 days
H2O cluster name:	ubuntu
H2O cluster total nodes:	1
H2O cluster free memory:	8.86 Gb
H2O cluster total cores:	8
H2O cluster allowed cores:	8
H2O cluster status:	locked, healthy
H2O connection url:	http://localhost:54321
H2O connection proxy:	None
Python version:	2.7.12 final

Then we make sure that the H2O cluster has the DeepWater distribution

In [2]:
```
from h2o.estimators.deepwater import H2ODeepWaterEstimator
if not H2ODeepWaterEstimator.available(): exit
```

H2O.ai

Configuration

Set the path to your h2o installation and download the 'bigdata' dataset using `./gradlew syncBigdataLaptop` from the H2O source distribution.

In [5]:
```
H2O_PATH=os.path.expanduser("~/h2o-3/")
```

Image Classification Task

H2O DeepWater allows you to specify a list of URIs (file paths) or URLs (links) to images, together with a response column (either a class membership (enum) or regression target (numeric)).

For this example, we use a small dataset that has a few hundred images, and three classes: cat, dog and mouse.

In [6]:
```
frame = h2o.import_file(H2O_PATH + "/bigdata/laptop/deepwater/imagenet/cat_dog_mouse.csv")
print(frame.dim)
print(frame.head(5))
```

```
Parse progress: |████████████████████████████████████████████████████████| 100%
[267, 2]
```

C1	C2
bigdata/laptop/deepwater/imagenet/cat/102194502_49f003abd9.jpg	cat
bigdata/laptop/deepwater/imagenet/cat/11146807_00a5f35255.jpg	cat
bigdata/laptop/deepwater/imagenet/cat/1140846215_70e326f868.jpg	cat
bigdata/laptop/deepwater/imagenet/cat/114170569_6cbdf4bbdb.jpg	cat
bigdata/laptop/deepwater/imagenet/cat/1217664848_de4c7fc296.jpg	cat

Deep Water Basic Usage

Unified Interface for TF, MXNet and Caffe

Choosing different network structures

```
model = H2ODeepWaterEstimator(epochs      = 500,
                              network     = "lenet",
                              image_shape = [28,28],  ## provide image size
                              channels    = 3,
                              backend     = "tensorflow",
                              model_id    = "deepwater_tf_simple")

model.train(x = [0], # file path e.g. xxx/xxx/xxx.jpg
            y = 1, # label cat/dog/mouse
            training_frame = frame)

model.show()
```

```
deepwater Model Build progress: |████████████████████████████████████████| 100%
Model Details
=============
H2ODeepWaterEstimator :  Deep Water
Model Key:  deepwater_tf_simple
```

Change backend to “mxnet”, “caffe” or “auto”

Home | (Busy) !TensorFlow_Par | 54.158.0.123/notebooks/!TensorFlow_Paris_Demo.ipynb

jupyter !TensorFlow_Paris_Demo Last Checkpoint: 13 minutes ago (u

File Edit View Insert Cell Kernel Help

Markdown CellToolbar

```
print(frame.head(5))
```

Parse progress: |

[267, 2]

C1
bigdata/laptop/deepwater/imagenet/cat/102194502_49f003abd9.jpg
bigdata/laptop/deepwater/imagenet/cat/11146807_00a5f35255.jpg
bigdata/laptop/deepwater/imagenet/cat/1140846215_70e326f868.jpg
bigdata/laptop/deepwater/imagenet/cat/114170569_6cbdf4bbdb.jpg
bigdata/laptop/deepwater/imagenet/cat/1217664848_de4c7fc296.jpg

ubuntu@ip-10-164-48-74: ~

ubuntu@ip-10-164-48-74: ~ 78x9

```
Every 2.0s: gpustat -cp                          Wed Nov 30 09:37:01 2016

ip-10-164-48-74  Wed Nov 30 09:37:01 2016
[0] GRID K520         | 34'C,  76 % |  3806 /  4036 MB | java/1357(3804M)
```

Using GPU for training

ubuntu@ip-10-164-48-74: ~ 78x9

```
  1  [|||||                                  15.2%]   5  [|||||                                  10.8%]
  2  [|||||||||                              26.2%]   6  [|||||                                  11.0%]
  3  [|||||                                  14.0%]   7  [|||||                                  10.5%]
  4  [|||||                                  11.1%]   8  [|||||                                  14.5%]
  Mem[||||||||||||||||||9.09G/14.7G]   Tasks: 50, 122 thr; 2 running
  Swp[                          0K/0K]   Load average: 0.33 0.21 0.15
                                         Uptime: 06:54:06
```

To build a LeNet image classification model in H2O, simply specify `network = "lenet"` and the **Tensorflow** backend to use the tensorflow lenet implementation:

In [*]:

```
model = H2ODeepWaterEstimator(epochs      = 500,
                              network     = "lenet",
                              image_shape = [28,28],  ## provide image size
                              channels    = 3,
                              backend     = "tensorflow",
                              model_id    = "deepwater_tf_simple")

model.train(x = [0], # file path e.g. xxx/xxx/xxx.jpg
            y = 1, # label cat/dog/mouse
            training_frame = frame)

model.show()
```

deepwater Model Build progress: |

H2O.ai

To build a LeNet image classification model in H2O, simply specify `network = "lenet"` and the **Tensorflow** backend to use the tensorflow lenet implementation:

In [12]:
```
model = H2ODeepWaterEstimator(epochs      = 500,
                              network     = "lenet",
                              image_shape = [28,28],  ## provide image size
                              channels    = 3,
                              backend     = "tensorflow",
                              model_id    = "deepwater_tf_simple")

model.train(x = [0], # file path e.g. xxx/xxx/xxx.jpg
            y = 1, # label cat/dog/mouse
            training_frame = frame)

model.show()
```

```
deepwater Model Build progress: |████████████████████████████████████████| 100%
Model Details
=============
H2ODeepWaterEstimator :  Deep Water
Model Key:  deepwater_tf_simple


ModelMetricsMultinomial: deepwater
** Reported on train data. **

MSE: 0.177650603738
RMSE: 0.421486184517
LogLoss: 0.865899719937
Mean Per-Class Error: 0.217708629345
Confusion Matrix: vertical: actual; across: predicted
```

cat	dog	mouse	Error	Rate
85.0	2.0	3.0	0.0555556	5 / 90
18.0	61.0	6.0	0.2823529	24 / 85
27.0	2.0	63.0	0.3152174	29 / 92
130.0	65.0	72.0	0.2172285	58 / 267

Deep Water – Custom Network

If you'd like to build your own Tensorflow network architecture, then this is easy as well. In this example script, we are using the **Tensorflow** backend. Models can easily be imported/exported between H2O and Tensorflow since H2O uses Tensorflow's format for model definition.

In [8]:
```
def simple_model(w, h, channels, classes):
    import json
    import tensorflow as tf
    # always create a new graph inside ipython or
    # the default one will be used and can lead to
    # unexpected behavior
    graph = tf.Graph()
    with graph.as_default():
        size = w * h * channels
        x = tf.placeholder(tf.float32, [None, size])
        W = tf.Variable(tf.zeros([size, classes]))
        b = tf.Variable(tf.zeros([classes]))
        y = tf.matmul(x, W) + b

        # labels
        y_ = tf.placeholder(tf.float32, [None, classes])

        # accuracy
        correct_prediction = tf.equal(tf.argmax(y, 1),
                                      tf.argmax(y_, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

        # train
        cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
        train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

        tf.add_to_collection("train", train_step)
        # this is required by the h2o tensorflow backend
        global_step = tf.Variable(0, name="global_step", trainable=False)

        init = tf.initialize_all_variables()
        tf.add_to_collection("init", init)
        tf.add_to_collection("logits", y)
        saver = tf.train.Saver()
        meta = json.dumps({
                "inputs": {"batch_image_input": x.name, "categorical_labels": y_.name},
                "outputs": {"categorical_logits": y.name},
                "metrics": {"accuracy": accuracy.name, "total_loss": cross_entropy.name},
                "parameters": {"global_step": global_step.name},
        })
        print(meta)
        tf.add_to_collection("meta", meta)
        filename = "/tmp/lenet_tensorflow.meta"
        tf.train.export_meta_graph(filename, saver_def=saver.as_saver_def())
    return filename
```

Saving the custom network structure as a file

H2O.ai

In [13]:
```
model = H2ODeepWaterEstimator(epochs                 = 500,
                              network_definition_file = filename, ## specify the model
                              image_shape             = [28,28],  ## provide expected image size
                              channels                = 3,
                              backend                 = "tensorflow",
                              model_id                = "deepwater_tf_custom")

model.train(x = [0], # file path e.g. xxx/xxx/xxx.jpg
            y = 1, # label cat/dog/mouse
            training_frame = frame)

model.show()
```

Specifying the custom network structure for training

```
deepwater Model Build progress: |████████████████████████████████████████| 100%
Model Details
=============
H2ODeepWaterEstimator :  Deep Water
Model Key:  deepwater_tf_custom


ModelMetricsMultinomial: deepwater
** Reported on train data. **

MSE: 6.60075876885e+12
RMSE: 2569194.18668
LogLoss: -14.4921790248
Mean Per-Class Error: 0.0
Confusion Matrix: vertical: actual; across: predicted
```

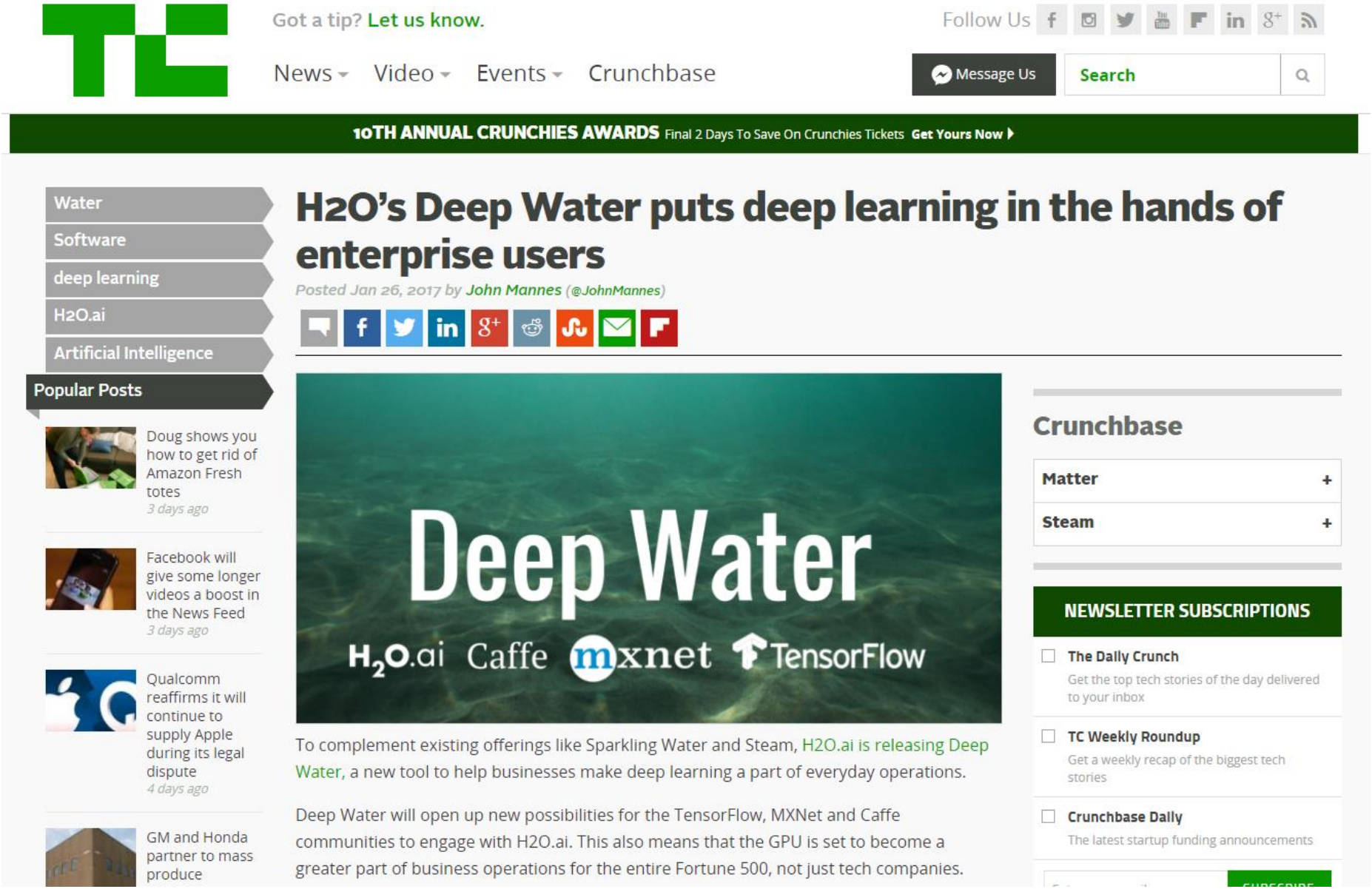
cat	dog	mouse	Error	Rate
90.0	0.0	0.0	0.0	0 / 90
0.0	85.0	0.0	0.0	0 / 85
0.0	0.0	92.0	0.0	0 / 92
90.0	85.0	92.0	0.0	0 / 267

Conclusions

Project “Deep Water”

- H_2O + TF + MXNet + Caffe
  - a powerful combination of widely used open source machine learning libraries.
- All Goodies from H_2O
  - inherits all H_2O properties in scalability, ease of use and deployment.
- Unified Interface
  - allows users to build, stack and deploy deep learning models from different libraries efficiently.

- 100% Open Source
  - the party will get bigger!



https://techcrunch.com/2017/01/26/h2os-deep-water-puts-deep-learning-in-the-hands-of-enterprise-users/

H2O.ai

Recap

Learning Objectives

- Understand the basic usage of H2O deep learning.
- Use H_2O deep autoencoder for outlier detection.

- (Optional) Introduction to Deep Water
  - Next-Gen H_2O Deep Learning

Thanks!

- Slides
  - bit.ly/h2o_meetups
- Contact
  - joe@h2o.ai
  - @matlabulous
  - github.com/woobe

H2O.ai

Making Machine Learning Accessible to Everyone

Photo credit: Virgin Media