Exploring and Evaluating Big-data Machine Learning Frameworks

Harsha H. Chunduri

CSP 554 Big Data Technologies

Illinois Institute of Technology

Illinois Institute of Technology

hchunduri@hawk.iit.edu

Abstract:
A brief discussion and comparison of some of the most common machine learning frameworks/libraries are presented in this paper. Some code comparisons, distinguishing features and key observations made while exploring these frameworks are discussed in depth, and the advantages and drawbacks of each are presented. Frameworks that are evaluated: Tensorflow (keras), Pytorch, H2O, MLlib and Deequ. The first few frameworks are commonly used for Machine Learning, while Deequ is a library designed to assess data quality at scale.
Keywords: API, DL, ML, Frameworks, Libraries

I. Introduction:

Machine learning has become the de-facto solution to address problems with data. With 2.5 quintillion bytes of data generated everyday, the capability to leverage and improve products with the help of machine learning is significantly increasing. Data is becoming the new currency[1]. Many organizations have come up with state-of-the-art libraries to solve problems related to Machine Learning (ML), Deep Learning (DL), neural networks, etc. Tensorflow, developed by Google is one of the popular DL frameworks used by data scientists all over the world. Tensorflow can be executed on different platforms ranging from massively parallel clusters to cell phones[2]. Tensorflow is primarily used for Deep Neural Networks (DNN) in image classification, object recognition and Natural Language Processing (NLP). Tensorflow supports python, Java and C++. But, there are many other open source contributions to support in other languages like Go, scala, etc.

Another open source library which is popular for distributed machine learning applications is Apache Spark's MLlib. MLlib has many features for extracting features, reducing the dimensionality and also has support to build ML pipelines. It supports two different API's, one for the traditional Resilient Distributed Datasets (RDD) and another for the DataFrame. MLlib supports different classification, regression, decision trees and many other algorithms at scale[3]. Spark's Application Programming Interface (API) supports scala, java, python and R languages.

Facebook's research lab of AI developed an open source library to put ML to good use in the areas of computer vision (CV) and NLP. There is a constant trade-off between speed and ease-of-use when applying machine learning libraries, pytorch addresses these issues by providing the code as a model making it easier to debug the model while performing executions without compromising on efficiency[4]. Pytorch is highly optimized especially for use in the field of DL on top of CPUs and GPUs.

Using tensorflow, a high level API called Keras was developed by an engineer François Chollet in python for building machine learning models, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). Keras runs on top of the popular TensorFlow. The power of keras comes from the fact that it is easy to use with python. Keras simply put is a wrapper around TensorFlow and Theano[5].

H2O, yet another open source framework for ML applications, has API's in R, Scala, Java and Python, similar to MLlib. H2O's key theme is the in-memory compression technique which helps handle millions of rows in the memory on a small sized system. H2O is fast, and also runs on Hadoop and Spark[6]. Algorithms include supervised and unsupervised learning methods such as random forest, XGBoost, Word2vec to name some.

While machine learning frameworks are popular with data, the quality of the data is one aspect which affects the overall performance of the outputs. As the conceptual saying goes "Garbage in, garbage out", incorrect data or data having a lot of missing values will result in bad performance of the models. Developers at Amazon have come up with a data quality check framework to automate data quality checks. Deequ conducts tests for the data to verify the quality of data. Deequ is a library on top of Spark's architecture, that makes it easier to scale to large datasets efficiently. Deequ allows developers to define the requirements in declarative format. This framework helps the developers and operations team to automate the entire data validation process and notify about data that seems off. Tests integration with the data is also a prime feature of deequ and helps validate the quality of data at hand[7-8].

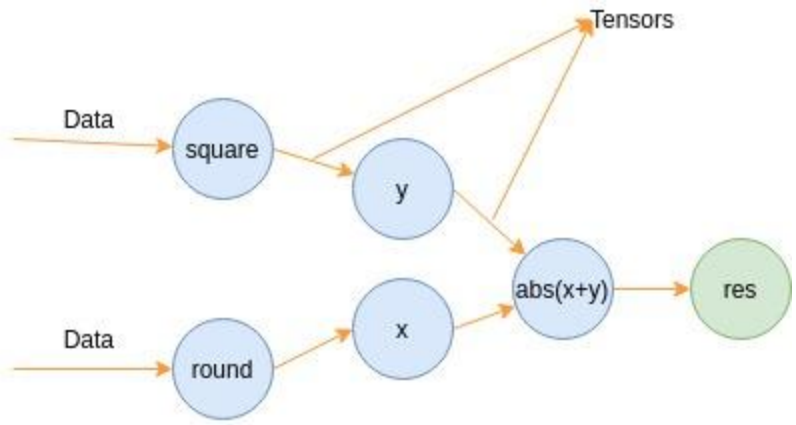
The frameworks that are mentioned in this article are generalized as machine learning frameworks. But, almost all of them have support for deep learning applications as well. Majority of the machine learning applications are supported by scikit-learn[9].

A question that usually pops up is why are there so many libraries out there for ML and DL applications. The field of machine learning is growing a lot and it still hasn't reached the depth scientists would like to and the experimentations differ from case to case. With data generating at an unprecedented pace, the amount of computing resources required to process are increasing. As a result, many organizations came up with different libraries as part of their research to process data. Experimentation, usability and refinement are continuous processes and many more libraries might also come in the near future with better approaches. Mahout, MLlib, H2O and SAMOA libraries are studied with respect to the big data setting[10].

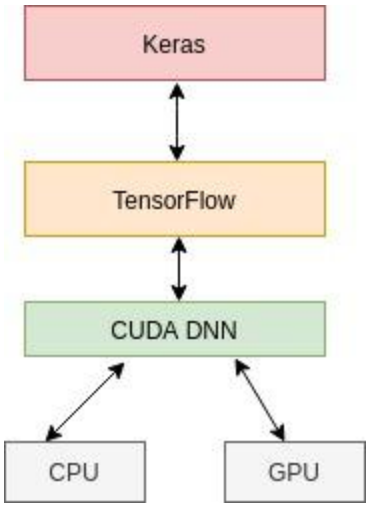
However, every library is unique in its own way. In this report, we will try to explore the libraries in a more detailed fashion and also compare different architectures and the underlying advantages and usability of the libraries in different cases.

II. Tensorflow and Keras:

TensorFlow can be described as a complex yet optimized amalgamation of machine learning algorithms and neural networks represented in computational directed graphs. Every aspect in tensorflow is generally represented as a graph.



A simple equation res = abs(x+y) can be represented as a graph shown above. The directed arrows (edges) are called Tensors and data flows through the edges doing computations such as round, absolute, etc. Hence the name TensorFlow. Tensorflow is highly used for building binary image classification models and neural networks.



As Keras is a high level API, on top of TensorFlow, it offers more modularity compared to direct implementation in tensorflow. Let us understand with the help of an example. One of the most commonly used functions in keras is the "*Conv2D*"[11], shown in the image below.

```
class Conv2D(Conv):
  def __init__(self, filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1),
               groups=1, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros',
               kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
               bias_constraint=None, **kwargs):
    super(Conv2D, self).__init__(rank=2, filters=filters, kernel_size=kernel_size, strides=strides, padding=padding,
        data_format=data_format, dilation_rate=dilation_rate, groups=groups, activation=activations.get(activation),
        use_bias=use_bias, kernel_initializer=initializers.get(kernel_initializer), bias_initializer=initializers.get(bias_initializer),
        kernel_regularizer=regularizers.get(kernel_regularizer), bias_regularizer=regularizers.get(bias_regularizer),
        activity_regularizer=regularizers.get(activity_regularizer), kernel_constraint=constraints.get(kernel_constraint),
        bias_constraint=constraints.get(bias_constraint), **kwargs)
```

Convolutions in images are used to process the images for various reasons such as edge detecting, sharpening, increasing the contrast of images, etc. This function is a 2 dimensional convolutional layer taking different parameters. We can see that the class Conv2D takes in a Convolution "*Conv"* and also different parameters such as filters, stride sizes, padding, etcetera are intialized. Keras, as mentioned, runs on top of tensorflow or more precisely a wrapper around tensorflow. While passing the images to a convolutional layer, keras makes it easier to change parameters instead of writing entirely from scratch with tensorflow. This helps avoid the programmer to deep dive into the convolutional granularity and can simply get us started with the modelling part without much trouble.

However, the same function can also be written in tensorflow as follows with some helper libraries of tensorflow.

```
1. def conv2d(input_to_be_taken, filtr, strd_size):
2.     out = tf.nn.conv2d(input_to_be_taken, filtr, strides=[1, strd_size, strd_size, 1],
       padding=0.3)
3.     return tf.nn.leaky_relu(out, alpha=0.092)
```

In the snippet above, an input, filter and the respective stride sizes of an image are taken and the output is the convoluted two dimensional image with changed alpha and Rectified Linear Unit Activation Function (RELU). (RELU is an activation function used when the inputs' gradients are below the "0" which causes the neurons to paralyze or die).
The functionality with the help of tensorflow's neural network library offers "*conv2d"* functionality. The snippet for the same is taken from the library[12] and is shown below.

```
tf.nn.conv2d code:

def conv2d_v2(input, filters, strides, padding, data_format="NHWC", dilations=None, name=None):
    return conv2d(input, filters, strides, padding, use_cudnn_on_gpu=True, data_format=data_format, dilations=dilations, name=name)
def leaky_relu(features, alpha=0.2, name=None):
    with ops.name_scope(name, "LeakyRelu", [features, alpha]) as name:
    features = ops.convert_to_tensor(features, name="features")
    if features.dtype.is_integer:
      features = math_ops.cast(features, dtypes.float32)
    if isinstance(alpha, np.ndarray):
      alpha = alpha.item()
    return gen_nn_ops.leaky_relu(features, alpha=alpha, name=name)
```

Comparing the wrapper class that is written in keras with the granular code written with the snippet (and with the help of tensorflow's neural network (nn) functionality), we can observe that

1. Same functionality of keras can be produced in Tensorflow with much more granularity when compared to the wrapper class of keras.
2. This makes programmers fall into two categories. In order to get started quickly with the problem at hand, keras provide modularity of code whereas when a programmer wants to change the nitty-gritties of convolution, tensorflow can be used.
3. TensorFlow gives a lot of control over the parameters whereas keras assumes a certain level of abstraction.
4. Flexibility of changing is not high in keras in the works related to researching and changing parameters.
5. TensorFlow offers low level control such as using Cuda GPU (Nvidia), dilations, etc whereas keras doesn't offer them by default.
6. Higher the parameters passed to the model, higher the networks internally which leads to more epoch time.

III. Pytorch

One of the most significant things while using Pytorch is that it automatically takes a Graphics Processing Unity (GPU) if available or else just uses the Central Processing Unit (CPU). Pytorch makes it easier to understand if the GPU is available or not just by two lines of code.

```
[38] device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
     print(device)

     cuda:0
```

```
[39] torch.cuda.is_available()

     True
```

Cuda is an acronym for Compute Unified Device Architecture which acts as an API for driver. The function is_available() in the torch library returns a boolean value "True" if there is a GPU available.

```
[122] device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
      print(device)

      cpu
```

```
torch.cuda.is_available()

False
```

If there is no GPU available this function returns “False” as a result. The above code is executed by taking the Google Colab Notebook[13] as toggling from CPU to GPU is free and easier.

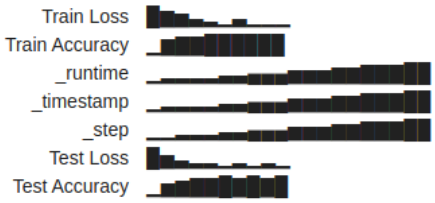
Run summary:

Train Loss	0.07489
Train Accuracy	97.02
_runtime	20
_timestamp	1620603506
_step	19
Test Loss	921.57014
Test Accuracy	97.09

Run history:

Train Loss	0.05775
Train Accuracy	96.43
_runtime	120
_timestamp	1620608323
_step	19
Test Loss	857.10682
Test Accuracy	97.29

Run history:



A sample code was run on the MNIST dataset[14]. The accuracy results are displayed above. It can be seen that the runtime in case of GPU is 20 whereas in the case of CPU is 120. No additional code is required to run the program as torch takes GPU by default if available.

```
def conv2d(input: Tensor, weight: Tensor, bias: Optional[Tensor]=None,
    stride: Union[_int, _size]=1, padding: Union[_int, _size]=0,
    dilation: Union[_int, _size]=1, groups: _int=1) -> Tensor
```

The conv2d function in Pytorch results in a “Tensor” as an output and takes parameters similar to that of Tensorflow except that CUDA is taken automatically whereas in tensorflow the GPU parameter must be defined exclusively.
Parallelism can be achieved with the help of DataParallel() function composed as follows.

```
def data_parallel(module, inputs, device_ids=None, output_device=None, dim=0, module_kwargs=None):
    if device_ids is None:
        device_ids = _get_all_device_indices()
    if output_device is None:
        output_device = device_ids[0]

    device_ids = [_get_device_index(x, True) for x in device_ids]
    output_device = _get_device_index(output_device, True)
    if len(device_ids) == 1:
        return module(*inputs[0], **module_kwargs[0])
    used_device_ids = device_ids[:len(inputs)]
    replicas = replicate(module, used_device_ids)
    outputs = parallel_apply(replicas, inputs, module_kwargs, used_device_ids)
    return gather(outputs, output_device, dim)
```

Key observations:

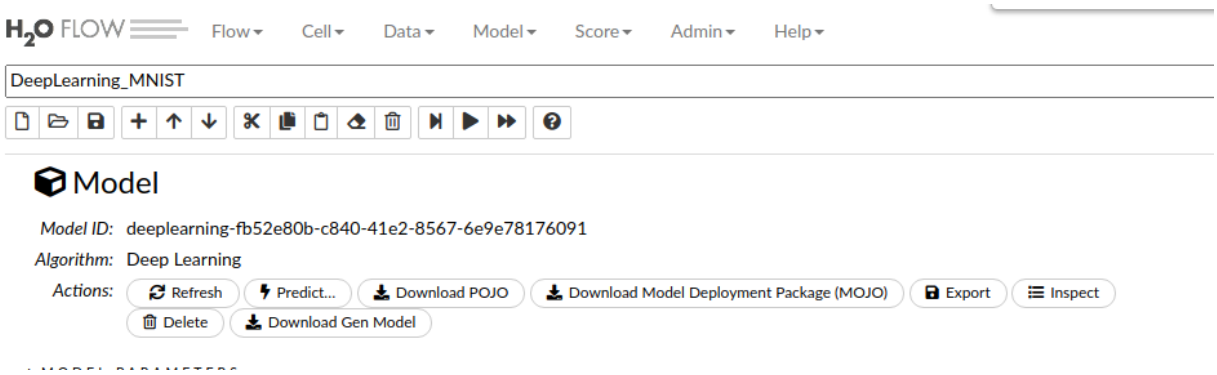
1. GPU need not be given as a parameter in Pytorch

2. Boilerplate code is wrapped up neatly in order for better usage in both tensorflow and pytrch.
3. Pytorch supports parallelism with the help of a wrapper function **torch.nn.DataParallel()** which will help parallelize a module by just passing it through this function.
4. Also, pytorch has cloud support where instances can be created directly from the code, and the output such as pickle files can be saved in the remote instances.

IV. H2O

H2O is a major player in the machine learning industry. H2O is also an open source framework offering in-memory implementation of various ML libraries. One of the most distinguishing features of H2O over other frameworks however, is the ability to scale on to a huge number of distributed systems with other available frameworks. Embedding the trained machine learning models into another language is extremely time consuming. Most of the latest machine learning algorithms or libraries are built on python. However, embedding these algorithms into a java based web application is quite a challenging task. H2O offers a functionality which supports building a Plain Old Java Object (POJO)[15]. These objects can be integrated with java based applications. Some other libraries which support this kind of functionality is the scala language. Scala and Java libraries can be used interchangeably as both these languages run on the Java Virtual Machine (JVM).

H2O Flow gives users an interface to communicate easily. It offers tremendous flexibility in terms of choosing parameters without having to remember parameters as the parameters can be set from a user interface (UI). In the below screenshot, it is evident that the H2O flow has eased the process of manually searching for optional commands such as predicting or creating jars for the model that is built. I have taken an example from the help section available in the H2O Flow.



Apart from the regular training process, H2O also has the CPU usage displays and network testing displays which are vital while training a model.

Machine learning engineers test their data on a lot of models in an attempt to increase the accuracy of the outputs. H2O has an “AutoML” feature, where multiple models can be trained and H2O displays them in the order of their ranking as to which model performs better over

other models. This saves developers' time to a great extent. Automating the mundane data science process is a key takeaway.
Key observations:

1. Limited deep learning algorithms
2. Highly scalable on the clusters
3. Conversion of python data frames to H2O dataframes
4. H2O's Sparkling water has support for the spark API where data pre-processing can be done in spark while H2O is used to train the model and scale on to machines with help of Spark(hence the name "sparkling" water[16]). Model prediction can also be done through H2O.
5. Exporting and integrating models with POJO onto java applications seamlessly.

V. MLlib:

Spark's MLlib is one of the four important APIs sitting on top of Spark core. The other three being spark streaming, GraphX and SparkSQL[17]. There are many algorithms that are currently being supported by spark. When it comes to big data analytics, the term "big" doesn't conform to a certain standard as data is ever growing. In this case, scaling to hundreds or even thousands of clusters is the only way to store or process it. The same is true for machine learning when applied on massive chunks of data.

There are two API's that used to be supported by Spark, one being the resilient distributed dataset based API and the other being a data frame based API. In the recent versions of the Spark, the data frame based API is the primary API in the spark MLlib package[18]. The core parts of Spark MLlib can be described as follows.

Statistics/ Utils	Engineering/Pipeline	Features	ML Algorithms
Correlation, Math	Persistence	Transformations	Clustering, Regression
Hypothesis Testing, Data handling	Evaluator, Model Selection, Tuning	Extractions, Dimensionality reduction	Filtering, classification

In-memory computation is the prime reason the computations are much faster in the Spark compared to other libraries. Of the above discussed frameworks, MLlib is the only library which doesn't offer official deep learning compatibility. But, because Spark is an open source framework, several deep learning algorithms and pipelines are being built. One of the pipelines that are actively used by the spark community is the deep learning pipeline by databricks[19].

The power of spark is utilised in an efficient way when deep learning is not (yet) supported by MLlib. Spark's python API can be used in tandem with tensorflow or keras and can be scaled up

to work in a clustered environment. High processing power and fault tolerant framework like MLlib is very important to implement machine learning algorithms as data can be distributed. However, one concerning thing that exists with the Mllib is the latency. Although high speeds are possible, network latency is something that has to be taken into account while dealing with huge chunks.

Observations while evaluating MLlib:

1. Can work with python's ML and DL libraries which compensates for lack of DL support
2. MLlib has several algorithms in classification, regression, filtering and support vector machines[18]. Random forest and tree classifications are used highly
3. Scalable machine learning is possible with the help of scala.
4. Open source deep learning support
5. Primary API is the dataframe API
6. Good with iterations and hence much practical to data science applications

VI. Deequ

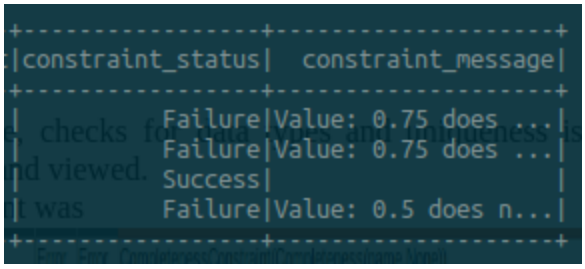
While evaluating the machine learning frameworks were important, there is a growing need to address the quality of the data[20]. As the data demands increase, unit tests to understand and keeping the quality of data in check is important. Irrelevant, improper or missing data is often the cause for bad performance on machine learning algorithms leading to low quality decision making[21]. As in software, to check the functionality of the code written, unit tests are performed. In the same way, deequ can test for the quality of data, schema of the data, etcetera. This can be done using SparkSQL queries. The quality of the data can be assessed with the help of Deequ. Dataframe based tables can be detected and surveyed for quality with the help of AWS' Deequ.
Taking a sample csv as follows:

id	name	gender
1	john doe	M
2	Jane doe	F
3	dove	T
2		

Here, the column "id" contains a duplicate
Following code was written in scala to check the validity of this csv. This csv was loaded as a dataframe.

```
import com.amazon.deequ.{VerificationSuite, VerificationResult}
import com.amazon.deequ.VerificationResult.checkResultsAsDataFrame
import com.amazon.deequ.checks.{Check, CheckLevel}
import com.amazon.deequ.constraints.ConstrainableDataTypes
val _check = Check(CheckLevel.Error, "Data Validation Check").isComplete("name")
  .isComplete("gender").hasDataType("id", ConstrainableDataTypes.Integral)
  .isUnique("id")
val verificationResult: VerificationResult = { VerificationSuite().onData(df).addCheck(_check).run()}
val resultDataFrame = checkResultsAsDataFrame(spark, verificationResult)
```

In the code, checks for data types and uniqueness is given. And the output is stored as a dataframe and viewed. Success doesn't have a constraint message whereas other columns failing will give what constraint is failed to get captured.



```
+-----------------+--------------------+
|constraint_status|  constraint_message|
+-----------------+--------------------+
|          Failure|Value: 0.75 does ...|
|          Failure|Value: 0.75 does ...|
|          Success|                    |
|          Failure|Value: 0.5 does n...|
+-----------------+--------------------+
```

The resultant was converted into a csv and is as follows

Data Validation Check	Error	Error	CompletenessConstraint(Completeness(name,None))	Failure	Value: 0.75 does not meet the constraint requirement!
Data Validation Check	Error	Error	CompletenessConstraint(Completeness(gender,None))	Failure	Value: 0.75 does not meet the constraint requirement!
Data Validation Check	Error	Error	AnalysisBasedConstraint(DataType(id,None),<function1>,Some(<function1>),None)	Success	
Data Validation Check	Error	Error	UniquenessConstraint(Uniqueness(List(id),None))	Failure	Value: 0.5 does not meet the constraint requirement!

It can be seen that in the last row, the uniqueness was marked as "Failure".
This was a simple way to unit test the dataframe using scala and spark. There are many functions to check regular expressions, dates, numbers, percentages, strings and many more.

In the similar way, data can be checked for nulls, missing values and metrics of missing values can also be gathered. Deequ is a powerful tool to assess the quality of data before starting a machine learning or deep learning project.

Some of the key observations:

1. Can work with Scala-spark and Pyspark as well.
2. Easy to work with package
3. Version issues when using latest versions of the Deequ package. Must check in the maven before starting to work
4. Metrics and output data frames can be asserted for quality checks

VII. Conclusion

Although there are different use cases for all the above libraries, the ability to work on massive datasets without having to configure is the key takeaway for every framework. The importance of data quality should be equally important before implementing a machine learning project. In this paper, we have seen machine learning and/or deep learning frameworks such as Tensorflow with keras, Pytorch, H2O, MLlib and finally a data quality assessment tool called Deequ along with key observations for the same.

References:

[1]. Gates, Carrie, and Peter Matthews. "Data is the new currency." In Proceedings of the 2014 New Security Paradigms Workshop, pp. 105-116. 2014.

[2]. Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[3]. Meng, Xiangrui, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman et al. "Mllib: Machine learning in apache spark." The Journal of Machine Learning Research 17, no. 1 (2016): 1235-1241.

[4]. Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen et al. "Pytorch: An imperative style, high-performance deep learning library." arXiv preprint arXiv:1912.01703 (2019).

[5]. Arnold, Taylor B. "kerasR: R interface to the keras deep learning library." Journal of Open Source Software 2, no. 14 (2017): 296.

[6]. Aiello, Spencer, Eric Eckstrand, Anqi Fu, Mark Landry, and Patrick Aboyoun. "Machine Learning with R and H2O." H2O booklet 550 (2016).

[7]. Schelter, Sebastian, Felix Biessmann, Dustin Lange, Tammo Rukat, Phillipp Schmidt, Stephan Seufert, Pierre Brunelle, and Andrey Taptunov. "Unit testing data with deequ." In Proceedings of the 2019 International Conference on Management of Data, pp. 1993-1996. 2019.

[8]. https://github.com/awslabs/deequ.

[9]. Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[10]. Richter, Aaron N., Taghi M. Khoshgoftaar, Sara Landset, and Tawfiq Hasanin. "A multi-dimensional comparison of toolkits for machine learning with big data." In 2015 IEEE International Conference on Information Reuse and Integration, pp. 1-8. IEEE, 2015.

[11]. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

[12]. https://www.kite.com/python/docs/tensorflow.contrib.grid_rnn.nn.conv2d_v2

[13]. https://colab.research.google.com/notebooks/intro.ipynb

[14]. http://yann.lecun.com/exdb/mnist/

[15]. https://h2o-release.s3.amazonaws.com/h2o/rel-ueno/2/docs-website/h2o-docs/pojo-quick-start.html

[16]. Malohlava, Michal, Jakub Hava, and Nidhi Mehta. "Machine learning with sparkling water: H2o+ spark." H2O. ai Inc (2016).

[17]. https://spark.apache.org/

[18]. https://spark.apache.org/docs/latest/ml-guide.html

[19]. https://github.com/databricks/spark-deep-learning

[20]. Wang, Richard Y., and Diane M. Strong. "Beyond accuracy: What data quality means to data consumers." Journal of management information systems 12.4 (1996): 5-33.

[21]. Sessions, Valerie, and Marco Valtorta. "The Effects of Data Quality on Machine Learning Algorithms." ICIQ 6 (2006): 485-498.