**Chapter 31. Deep Learning**

Deep learning is one of the most exciting areas of development around Spark due to its ability to solve several previously difficult machine learning problems, especially those involving unstructured data such as images, audio, and text. This chapter will cover how Spark works in tandem with deep learning, and some of the different approaches you can use to work with Spark and deep learning together.

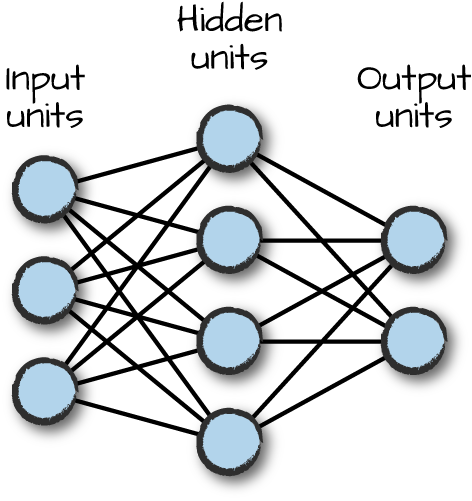
Because deep learning is still a new field, many of the newest tools are implemented in external libraries. This chapter will not focus on packages that are necessarily core to Spark but rather on the massive amount of innovation in libraries built on top of Spark. We will start with several high-level ways to use deep learning on Spark, discuss when to use each one, and then go over the libraries available for them. As usual, we will include end-to-end examples.

**NOTE**

To make the most of this chapter you should know at least the basics of deep learning as well as the basics of Spark. With that being said, we point to an excellent resource at the beginning of this part of the book called the [Deep Learning Book](http://www.deeplearningbook.org/), by some of the top researchers in this area.

**What Is Deep Learning?**

To define deep learning, we must first define neural networks. A neural network is a graph of nodes with weights and activation functions. These nodes are organized into *layers* that are stacked on top of one another. Each layer is connected, either partially or completely, to the previous layer in the network. By stacking layers one after the other, these simple functions can learn to recognize more and more complex signals in the input: simple lines with one layer, circles and squares with the next layer, complex textures in another, and finally the full object or output you hope to identify. The goal is to train the network to associate certain inputs with certain outputs by tuning the weights associated with each connection and the values of each node in the network. Figure 31-1 shows the simple neural network.



*Figure 31-1. A neural network*

*Deep learning*, or *deep neural networks*, stack many of these layers together into various different architectures. Neural networks themselves have existed for decades, and have waxed and waned in terms of popularity for various machine learning problems. Recently, however, a combination of much larger datasets (e.g., the ImageNet corpus for object recognition), powerful hardware (clusters and GPUs), and new training algorithms have enabled training much larger neural networks that outperform previous approaches in many machine learning tasks. Typical machine learning techniques typically cannot continue to perform well as more data is added; their performance hits a ceiling. Deep learning can benefit from enormous amounts of data and information and it is not uncommon for deep learning datasets to be orders of magnitude larger than other machine learning datasets. Deep neural networks have now become the standard in computer vision, speech processing, and some natural language tasks, where they often “learn” better features than previous hand-tuned models. They are also actively being applied in other areas of machine learning. Apache Spark’s strength as a big data and parallel computing system makes it a natural framework to use with deep learning.

Researchers and engineers have put a lot of effort into speeding up these neural network-like calculations. Nowadays, the most popular way to use neural networks or deep learning is to use a framework, implemented by a research institute or corporation. The most popular as of the time of this writing are TensorFlow, MXNet, Keras, and PyTorch. This area is rapidly evolving so it’s always worth searching around for others.

**Ways of Using Deep Learning in Spark**

For the most part, regardless of which application you are targeting, there are three major ways to use deep learning in Spark:

Inference

The simplest way to use deep learning is to take a pretrained model and apply it to large datasets in parallel using Spark. For example, you could use an image classification model, trained using a standard dataset like ImageNet, and apply it to your own image collection to identify pandas, flowers, or cars. Many organizations publish large, pretrained models on common datasets (e.g., Faster R-CNN and YOLO for object detection), so you can often take a model from your favorite deep learning framework and apply it in parallel using a Spark function. Using PySpark, you could simply call a framework such as TensorFlow or PyTorch in a map function to get distributed inference, though some of the libraries we discuss for it make further optimizations beyond simply calling these libraries in a map function.

Featurization and transfer learning

The next level of complexity is to use an existing model as a *featurizer* instead of taking its final output. Many deep learning models learn useful feature representations in their lower layers as they get trained for an end-to-end task. For example, a classifier trained on the ImageNet dataset will also learn low-level features present in all natural images, such as edges and textures. We can then use these features to learn models for a new problem not covered by the original dataset. This method is called *transfer learning*, and generally involves the last few layers of a pretrained model and retraining them with the data of interest. Transfer learning is also especially useful if you do not have a large amount of training data: training a full-blown network from scratch requires a dataset of hundreds of thousands of images, like ImageNet, to avoid overfitting, which will not be available in many business contexts. In contrast, transfer learning can work even with a few thousand images because it updates fewer parameters.

Model training

Spark can also be used to train a new deep learning model from scratch. There are two common methods here. First, you can use a Spark cluster to parallelize the training of a *single* model over multiple servers, communicating updates between them. Alternatively, some libraries let the user train *multiple* instances of similar models in parallel to try various model architectures and hyperparameters, accelerating the model search and tuning process. In both cases, Spark’s deep learning libraries make it simple to pass data from RDDs and DataFrames to deep learning algorithms. Finally, even if you do not wish to train your model in parallel, these libraries can be used to extract data from a cluster and export it to a single-machine training script using the native data format of frameworks like TensorFlow.

In all three cases, the deep learning code typically runs as part of a larger application that includes Extract, Transform, and Load (ETL) steps to parse the input data, I/O from various sources, and potentially batch or streaming inference. For these other parts of the application, you can simply use the DataFrame, RDD, and MLlib APIs described earlier in this book. One of Spark’s strengths is the ease of combining these steps into a single parallel workflow.

**Deep Learning Libraries**

In this section, we’ll survey a few of the most popular libraries available for deep learning in Spark. We will describe the main use cases of the library and link them to references or examples when possible. This list is not meant to be exhaustive, because the field is rapidly evolving. We encourage you to check each library’s website and the Spark documentation for the latest updates.

**MLlib Neural Network Support**

Spark’s MLlib currently has native support for a single deep learning algorithm: the ml.classification.MultilayerPerceptronClassifier class’s multilayer perceptron classifier. This class is limited to training relatively shallow networks containing fully connected layers with the sigmoid activation function and an output layer with a softmax activation function. This class is most useful for training the last few layers of a classification model when using transfer learning on top of an existing deep learning–based featurizer. For example, it can be added on top of the Deep Learning Pipelines library we describe later in this chapter to quickly perform transfer learning over Keras and TensorFlow models.

**TensorFrames**

[TensorFrames](https://github.com/databricks/tensorframes) is an inference and transfer learning-oriented library that makes it easy to pass data between Spark DataFrames and TensorFlow. It supports Python and Scala interfaces and focuses on providing a simple but optimized interface to pass data from TensorFlow to Spark and back. In particular, using TensorFrames to apply a model over Spark DataFrames is generally more efficient than calling a Python map function that directly invokes the TensorFlow model, due to faster data transfer and amortization of the startup cost. TensorFrames is most useful for inference, in both streaming and batch settings, and for transfer learning, where you can apply an existing model over raw data to featurize it, then learn the last layers using a MultilayerPerceptronClassifier or even a simpler logistic regression or random forest classifier over the data.

**BigDL**

[BigDL](https://github.com/intel-analytics/BigDL) is a distributed deep learning framework for Apache Spark primarily developed by Intel. It aims to support distributed training of large models as well as fast applications of these models using inference. One key advantage of BigDL over the other libraries described here is that it is primarily optimized to use CPUs instead of GPUs, making it efficient to run on an existing, CPU-based cluster (e.g., an Apache Hadoop deployment). BigDL provides high-level APIs to build neural networks from scratch and automatically distributes all operations by default. It can also train models described with the Keras DL library.

**TensorFlowOnSpark**

[TensorFlowOnSpark](https://github.com/yahoo/TensorFlowOnSpark) is a widely used library that can train TensorFlow models in a parallel fashion on Spark clusters. TensorFlow includes some foundations to do distributed training, but it still needs to rely on a cluster manager for managing the hardware and data communications. It does not come with a cluster manager or a distributed I/O layer out of the box. TensorFlowOnSpark launches TensorFlow’s existing distributed mode inside a Spark job, and automatically feeds data from Spark RDDs or DataFrames into the TensorFlow job. If you already know how to use TensorFlow’s distributed mode, TensorFlowOnSpark makes it easy to launch your job inside a Spark cluster and pass it data processed with other Spark libraries (e.g., DataFrame transformations) from any input source Spark supports. TensorFlowOnSpark was originally developed at Yahoo! and is also used in production at other large organizations. The project also integrates with Spark’s ML Pipelines API.

**DeepLearning4J**

[DeepLearning4j](https://deeplearning4j.org/spark) is an open-source, distributed deep learning project in Java and Scala that provides both single-node and distributed training options. One of its advantages over Python-based deep learning frameworks is that it was primarily designed for the JVM, making it more convenient for groups that do not wish to add Python to their development process. It includes a wide variety of training algorithms and support for CPUs as well as GPUs.

**Deep Learning Pipelines**

[Deep Learning Pipelines](https://github.com/databricks/spark-deep-learning) is an open source package from Databricks that integrates deep learning functionality into Spark’s ML Pipelines API. The package existing deep learning frameworks (TensorFlow and Keras at the time of writing), but focuses on two goals:

* Incorporating these frameworks into standard Spark APIs (such as ML Pipelines and Spark SQL) to make them very easy to use
* Distributing all computation by default

For example, Deep Learning Pipelines provides a DeepImageFeaturizer class that acts as a transformer in the Spark ML Pipeline API, allowing you to build a transfer learning pipeline in just a few lines of code (e.g., by adding a perceptron or logistic regression classifier on top). Likewise, the library supports parallel grid search over multiple model parameters using MLlib’s grid search and cross-validation API. Finally, users can export an ML model as a Spark SQL user-defined function and make it available to analysts using SQL or streaming applications. At the time of writing (summer 2017), Deep Learning Pipelines is under heavy development, so we encourage you to check its website for the latest updates.

Table 31-1 summarizes the various deep learning libraries and the main use cases they support:

*Table 31-1. Deep learning libraries*

|  |  |  |
| --- | --- | --- |
| **Library** | **Underlying DL framework** | **Use cases** |
| BigDL | BigDL | Distributed training, inference, ML Pipeline integration |
| DeepLearning4J | DeepLearning4J | Inference, transfer learning, distributed training |
| Deep Learning Pipelines | TensorFlow, Keras | Inference, transfer learning, multi-model training, ML Pipeline and Spark SQL integration |
| MLlib Perceptron | Spark | Distributed training, ML Pipeline integration |
| TensorFlowOnSpark | TensorFlow | Distributed training, ML Pipeline integration |
| TensorFrames | TensorFlow | Inference, transfer learning, DataFrame integration |

While there are several approaches different companies have taken to integrating Spark and deep learning libraries, the one currently aiming for the closest integration with MLlib and DataFrames is Deep Learning Pipelines. This library aims to improve Spark’s support for image and tensor data (which will be integrated into the core Spark codebase in Spark 2.3), and to make all deep learning functionality available in the ML Pipeline API. Its friendly API makes it the simplest way to run deep learning on Spark today and will be the focus of the remaining sections in this chapter.

**A Simple Example with Deep Learning Pipelines**

As we described, Deep Learning Pipelines provides high-level APIs for scalable deep learning by integrating popular deep learning frameworks with ML Pipelines and Spark SQL.

Deep Learning Pipelines builds on Spark’s ML Pipelines for training and on Spark DataFrames and SQL for deploying models. It includes high-level APIs for common aspects of deep learning so they can be done efficiently in a few lines of code:

* Working with images in Spark DataFrames;
* Applying deep learning models at scale, whether they are your own or standard popular models, to image and tensor data;
* Transfer learning using common pretrained deep learning models;
* Exporting models as Spark SQL functions to make it simple for all kinds of users to take advantage of deep learning; and
* Distributed deep learning hyperparameter tuning via ML Pipelines.

Deep Learning Pipelines currently only offers an API in Python, which is designed to work closely with existing Python deep learning packages such as TensorFlow and Keras.

**Setup**

[Deep Learning Pipelines](https://github.com/databricks/spark-deep-learning) is a Spark Package, so we’ll load it just like we loaded GraphFrames. Deep Learning Pipelines works on Spark 2.x and the package can be found [here](https://spark-packages.org/package/databricks/spark-deep-learning). You’re going to need to install a few Python dependencies, including [TensorFrames](https://spark-packages.org/package/databricks/tensorframes), [TensorFlow](https://www.tensorflow.org/), [Keras](https://keras.io/), and [h5py](http://www.h5py.org/). Make sure these are installed across both your driver and worker machines.

We’ll use the flowers dataset from the [TensorFlow retraining tutorial](https://www.tensorflow.org/tutorials/image_retraining). Now if you’re running this on a cluster of machines, you’re going to need a way to put these files on a distributed file system once you download them. We include a sample of these images in [the book’s GitHub Repository](https://github.com/databricks/Spark-The-Definitive-Guide).

**Images and DataFrames**

One of the historical challenges when working with images in Spark is that getting them into a DataFrame was difficult and tedious. Deep Learning Pipelines includes utility functions that make loading and decoding images in a distributed fashion easy. This is an area that’s changing rapidly. Currently, this is a part of Deep Learning Pipelines. Basic image loading and representation will be included in Spark 2.3. While it is not released yet, all of the examples in this chapter should be compatible with this upcoming version of Spark:

**from** **sparkdl** **import** readImages

img\_dir = '/data/deep-learning-images/'

image\_df = readImages(img\_dir)

The resulting DataFrame contains the path and then the image along with some associated metadata:

image\_df.printSchema()

root

|-- filePath: string (nullable = false)

|-- image: struct (nullable = true)

| |-- mode: string (nullable = false)

| |-- height: integer (nullable = false)

| |-- width: integer (nullable = false)

| |-- nChannels: integer (nullable = false)

| |-- data: binary (nullable = false)

**Transfer Learning**

Now that we have some data, we can get started with some simple transfer learning. Remember, this means leveraging a model that someone else created and modifying it to better suit our own purposes. First, we will load the data for each type of flower and create a training and test set:

**from** **sparkdl** **import** readImages

**from** **pyspark.sql.functions** **import** lit

tulips\_df = readImages(img\_dir + "/tulips").withColumn("label", lit(1))

daisy\_df = readImages(img\_dir + "/daisy").withColumn("label", lit(0))

tulips\_train, tulips\_test = tulips\_df.randomSplit([0.6, 0.4])

daisy\_train, daisy\_test = daisy\_df.randomSplit([0.6, 0.4])

train\_df = tulips\_train.unionAll(daisy\_train)

test\_df = tulips\_test.unionAll(daisy\_test)

In the next step we will leverage a transformer called the DeepImageFeaturizer. This will allow us to leverage a pretrained model called Inception, a powerful neural network successfully used to identify patterns in images. The version we are using is pretrained to work well with images of various common objects and animals. This is one of the standard pretrained models that ship with the Keras library. However, this particular neural network is not trained to recognize daisies and roses. So we’re going to use transfer learning in order to make it into something useful for our own purposes: distinguishing different flower types.

Note that we can use the same ML Pipeline concepts we learned about throughout this part of the book and leverage them with Deep Learning Pipelines: DeepImageFeaturizer is just an ML transformer. Additionally, all that we’ve done to extend this model is add on a logistic regression model in order to facilitate the training of our end model. We could use another classifier in its place. The following code snippet demonstrates adding this model (note this may take time to complete as it’s a fairly resource intensive process):

**from** **pyspark.ml.classification** **import** LogisticRegression

**from** **pyspark.ml** **import** Pipeline

**from** **sparkdl** **import** DeepImageFeaturizer

featurizer = DeepImageFeaturizer(inputCol="image", outputCol="features",

modelName="InceptionV3")

lr = LogisticRegression(maxIter=1, regParam=0.05, elasticNetParam=0.3,

labelCol="label")

p = Pipeline(stages=[featurizer, lr])

p\_model = p.fit(train\_df)

Once we’ve trained the model, we can use the same classification evaluator we used in [Chapter 25](https://www.safaribooksonline.com/library/view/spark-the-definitive/9781491912201/ch25.html#s6c2---preprocessing-and-feature-engineering). We can specify the metric we’d like to test and then evaluate it:

**from** **pyspark.ml.evaluation** **import** MulticlassClassificationEvaluator

tested\_df = p\_model.transform(test\_df)

evaluator = MulticlassClassificationEvaluator(metricName="accuracy")

**print**("Test set accuracy = " + str(evaluator.evaluate(tested\_df.select(

"prediction", "label"))))

With our DataFrame of examples, we can inspect the rows and images in which we made mistakes in the previous training:

**from** **pyspark.sql.types** **import** DoubleType

**from** **pyspark.sql.functions** **import** expr

*# a simple UDF to convert the value to a double*

**def** \_p1(v):

**return** float(v.array[1])

p1 = udf(\_p1, DoubleType())

df = tested\_df.withColumn("p\_1", p1(tested\_df.probability))

wrong\_df = df.orderBy(expr("abs(p\_1 - label)"), ascending=False)

wrong\_df.select("filePath", "p\_1", "label").limit(10).show()

**APPLYING DEEP LEARNING MODELS AT SCALE**

Spark DataFrames are a natural construct for applying deep learning models to a large-scale dataset. Deep Learning Pipelines provides a set of Transformers for applying TensorFlow graphs and TensorFlow-backed Keras models at scale. In addition, popular image models can be applied out of the box, without requiring any TensorFlow or Keras code. The transformers, backed by the Tensorframes library, efficiently handle the distribution of models and data to Spark tasks.

**Applying Popular Models**

There are many standard deep learning models for images. If the task at hand is very similar to what the models provide (e.g., object recognition with ImageNet classes), or merely for exploration, you can use the transformer DeepImagePredictor by simply specifying the model name. Deep Learning Pipelines supports a variety of standard models included in Keras, which are listed on its website. The following is an example of using DeepImagePredictor:

**from** **sparkdl** **import** readImages, DeepImagePredictor

image\_df = readImages(img\_dir)

predictor = DeepImagePredictor(

inputCol="image",

outputCol="predicted\_labels",

modelName="InceptionV3",

decodePredictions=True,

topK=10)

predictions\_df = predictor.transform(image\_df)

Notice that the predicted\_labels column shows “daisy” as a high probability class for all sample flowers using this base model. However, as can be seen from the differences in the probability values, the neural network has the information to discern the two flower types. As we can see, our transfer learning example was able to properly learn the differences between daisies and tulips starting from the base model:

df = p\_model.transform(image\_df)

**APPLYING CUSTOM KERAS MODELS**

Deep Learning Pipelines also allows us to apply a Keras model in a distributed manner using Spark. To do this, check the [user guide](http://bit.ly/2Edb6eQ) on the KerasImageFileTransformer. This loads a Keras model and applies it to a DataFrame column.

**APPLYING TENSORFLOW MODELS**

Deep Learning Pipelines, through its integration with TensorFlow, can be used to create custom transformers that manipulate images using TensorFlow. For instance, you could create a transformer to change the size of an image or modify the color spectrum. To do this, use the TFImageTransformer class.

**DEPLOYING MODELS AS SQL FUNCTIONS**

Another option is to deploy a model as a SQL function allowing any user who knows SQL to be able to use a deep learning model. Once this function is used, the resulting UDF function takes a column and produces the output of the particular model. For instance, you could apply Inception v3 to a variety of images by using the registerKeraImageUDF class:

**from** **keras.applications** **import** InceptionV3

**from** **sparkdl.udf.keras\_image\_model** **import** registerKerasImageUDF

**from** **keras.applications** **import** InceptionV3

registerKerasImageUDF("my\_keras\_inception\_udf", InceptionV3(weights="imagenet"))

This way, the power of deep learning is available to any Spark user, not just the specialist who built the model.

**Conclusion**

This chapter discussed several common approaches to using deep learning in Spark. We covered a variety of available libraries and then worked through some basic examples of common tasks. This area of Spark is under very active development and will continue to advance as time moves on so it’s worth checking in on the libraries to learn more as time goes on! Over time, the authors of this book hope to keep this chapter up to date with current developments.