Deep Learning Pipelines from Apache Spark

(Multi-Class Image Classification With Transfer Learning In PySpark)

Contents

- Using Keras (TensorFlow backend), PySpark, and **Deep Learning Pipelines** libraries to build an end-to-end deep learning solution for a multi-class image classification problem that runs on a Spark cluster.
- Spark is a robust open-source distributed analytics engine that can process large amounts of data with great speed.
- PySpark which is the python API for Spark that allows us to use Python programming language and leverage the power of Apache Spark.
- Using majorly **Deep Learning Pipelines(DLP)** which is a high-level Deep Learning framework that facilitates common Deep Learning workflows via the **Spark MLlib** Pipelines API. It currently supports TensorFlow and Keras with the TensorFlow-backend.
- We'll be using this **DLP** to build a multi-class image classifier that will run on the Spark cluster.

Plan

- integrate Deep Learning into the PySpark pipeline with the **DLP**
- Plan
  I. Transfer Learning: A short intuition of Deep Learning Pipelines (from Databricks)
  II. Data Set: Introducing the multiclass image data.
  III. Modeling: Build a model and training.
  IV. Evaluate: Evaluating the model performance using various evaluation metrics.

Transfer Learning

- Transfer learning is a technique in machine learning in general that focuses on saving knowledge (weights and biases) gained while solving one problem and further applying it to a different but related problem.
- **Deep Learning Pipelines** provide utilities to perform transfer learning on the images, which is one of the fastest ways to start using deep learning.
- With the concept of a ***Featurizer,*** Deep Learning Pipelines enables fast transfer learning on Spark-Cluster.
- Right now it provides the following neural nets for Transfer Learning:
  - InceptionV3
  - Xception
  - ResNet50
  - VGG16
  - VGG19

Example:

- The following example combines the **InceptionV3** model and **multinomial logistic regression** in Spark. A utility function from Deep Learning Pipelines called **DeepImageFeaturizer** automatically peels off the last layer of a pre-trained neural network and uses the output from all the previous layers as **features** for the logistic regression algorithm.
- Data Set
- The Data-set has ten numerical digits (graphemes or symbols indicating the numbers from 0 to 9). Numbers larger than 9 are hand-written using a positional base 10 numeral system choose randomly **50** images of each class.

- At first, we load all the images to SparkData Frame. Then we build the model and train it. After that, we'll evaluate the performance of our trained model.
- Load Images
- The data sets (from 0 to 9) contains almost 500 handwritten digits (50 images each class). Here we manually load each image into a **spark data-frame** with a target column. After loading the whole dataset we split into an **8:2** ratio randomly for the training set and final test set.
- Our goal is to train the model with the training data set and finally evaluate the model's performance with the test dataset.

```
# necessary import
from pyspark.sql import SparkSession
from pyspark.ml.image import ImageSchema
from pyspark.sql.functions import lit
from functools import reduce
# create a spark session
spark = SparkSession.builder.appName('DigitRecog').getOrCreate()
# loaded image
zero = ImageSchema.readImages("0").withColumn("label", lit(0))
one = ImageSchema.readImages("1").withColumn("label", lit(1))
two = ImageSchema.readImages("2").withColumn("label", lit(2))
three = ImageSchema.readImages("3").withColumn("label", lit(3))
four = ImageSchema.readImages("4").withColumn("label", lit(4))
five = ImageSchema.readImages("5").withColumn("label", lit(5))
six = ImageSchema.readImages("6").withColumn("label", lit(6))
seven = ImageSchema.readImages("7").withColumn("label", lit(7))
eight = ImageSchema.readImages("8").withColumn("label", lit(8))
nine = ImageSchema.readImages("9").withColumn("label", lit(9))
dataframes = [zero, one, two, three,four,
five, six, seven, eight, nine]
# merge data frame
df = reduce(lambda first, second: first.union(second), dataframes)# repartition dataframe
df = df.repartition(200)
# split the data-frame
train, test = df.randomSplit([0.8, 0.2], 42)
```

Model Training

- Here we combine the InceptionV3 model and logistic regression in Spark.
- The DeepImageFeaturizer automatically peels off the last layer of a pre-trained neural network and uses the output from all the previous layers as features for the logistic regression algorithm.
- Since logistic regression is a simple and fast algorithm, this transfer learning training can converge quickly.

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from sparkdl import DeepImageFeaturizer
# model: InceptionV3
# extracting feature from images
featurizer = DeepImageFeaturizer(inputCol="image",
outputCol="features",
modelName="InceptionV3")
# used as a multi class classifier
lr = LogisticRegression(maxIter=5, regParam=0.03,
elasticNetParam=0.5, labelCol="label")
# define a pipeline model
sparkdn = Pipeline(stages=[featurizer, lr])
spark_model = sparkdn.fit(train) # start fitting or training
```

Evaluation

- to evaluate model performance. We now like to evaluate four evaluation metrics such as **F1-score, Precision, Recall, Accuracy** on the test data set.

```
from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
# evaluate the model with test set
evaluator = MulticlassClassificationEvaluator()
tx_test = spark_model.transform(test)
print('F1-Score ', evaluator.evaluate(tx_test,
                                {evaluator.metricName: 'f1'}))
print('Precision ', evaluator.evaluate(tx_test,
                                 {evaluator.metricName:
'weightedPrecision'}))
print('Recall ', evaluator.evaluate(tx_test,
                               {evaluator.metricName: 'weightedRecall'}))
print('Accuracy ', evaluator.evaluate(tx_test,
                                {evaluator.metricName: 'accuracy'}))
```

```
F1-Score 0.8111782234361806
Precision 0.8422058244785519
Recall 0.8090909090909091
Accuracy 0.8090909090909091
```

Applying Deep Learning models at scale

- Deep Learning Pipelines supports running pre-trained models in a distributed manner with Spark, available in both batch and streaming data processing.
- It houses some of the most popular models, enabling users to start using deep learning without the costly step of training a model. The predictions of the model, of course, is done in parallel with all the benefits that come with Spark.
- In addition to using the built-in models, users can plug in Keras models and TensorFlow Graphs in a Spark prediction pipeline. This turns any single-node models on single-node tools into one that can be applied in a distributed fashion, on a large amount of data.

For Keras users

For applying Keras models in a distributed manner using Spark, KerasImageFileTransformer works on TensorFlow-backed Keras models.

•Internally creates a DataFrame containing a column of images by applying the user-specified image loading and processing function
•to the input DataFrame containing a column of image URIs
•Loads a Keras model from the given model file path
•Applies the model to the image DataFrame
To use the transformer, we first need to have a Keras model stored as a file. For this notebook we'll just save the Keras built-in InceptionV3
model instead of training one.

Working with general tensors

Deep Learning Pipelines also provides ways to apply models with tensor inputs (up to 2 dimensions),
written in popular deep learning libraries:
TensorFlow graphs
Keras models
The KerasTransformer applies a TensorFlow-backed Keras model to tensor inputs of up to 2 dimensions. It loads a Keras model from
a given model file path and applies the model to a column of arrays (where an array corresponds to a Tensor), outputting a column of arrays

Deploying Models in SQL

- One way to productionize a model is to deploy it as a Spark SQL User Defined Function, which allows anyone who knows SQL to use it. Deep Learning Pipelines provides mechanisms to take a deep learning model and *register* a Spark SQL User Defined Function (UDF). In particular, Deep Learning Pipelines 0.2.0 adds support for creating SQL UDFs from Keras models that work on image data.
- The resulting UDF takes a column (formatted as a image struct "SpImage") and produces the output of the given Keras model; e.g. for Inception V3, it produces a real valued score vector over the ImageNet object categories.