

Deep Learning Pipelines for Apache Spark on Databricks

Integrating Deep Learning with Spark for Scalable Applications

Arsiema Yohannes

Introduction

- **Deep Learning Pipelines (DLP):**
 - Library by Databricks for deep learning and transfer learning.
 - Integrates deep learning libraries with Spark MLlib and Spark SQL.
- **Objectives:**
 - High-level APIs for scalable deep learning.
 - Apply deep learning models on large datasets.
 - Enable transfer learning for new tasks.
- **Project Aim:**
 - Develop a robust pipeline for image datasets.
 - Implement transfer learning and evaluate scalability.

Key Technologies Used

- **Apache Spark:**
 - Distributed data processing and ML model training.
 - Scalability and efficiency for large datasets.
- **MLlib Pipelines:**
 - Building and tuning ML models with complex workflows.
- **Spark SQL:**
 - Query structured data and apply models directly in SQL queries.
- **Deep Learning Libraries:**
 - TensorFlow and Keras for model definition and training.



Design - Deep Learning Pipelines Tools

Image Data Handling:

- Tools for processing images in Spark DataFrames.

Transfer Learning:

- Adapt pre-trained models to new tasks.

Model Application at Scale:

- APIs for applying models to large datasets.

Deploying Models as SQL Functions:

- Use models in SQL queries.

Distributed Hyper-Parameter Tuning (Planned):

- Automated tuning with Spark MLlib Pipelines.

Implementation Overview

Image Data Handling:

- Setup cluster, install libraries and dependencies.
- Obtain and process image dataset.

Transfer Learning:

- Create training and test data frames.
- Train and evaluate models.

Applying Deep Learning Models at Scale:

- Use Spark MLlib Transformers for TensorFlow and Keras.
- Apply popular models and custom TensorFlow Graphs.

Implementation - Image Data Handling

Step 1: Setting Up the Cluster Environment

- Install `spark-deep-learning` and dependencies.

Step 2: Obtain Image Dataset

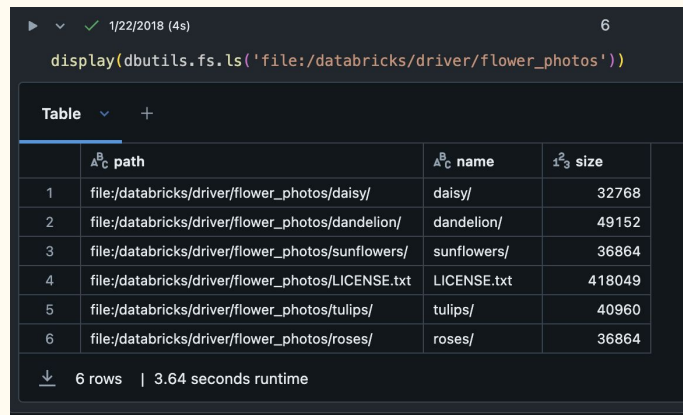
- Download and extract dataset.

Step 3: Create Sample Set

- Generate smaller set for demonstration.

Step 4: Load Images into DataFrames

- Use DLP tools to read images into Spark DataFrames.

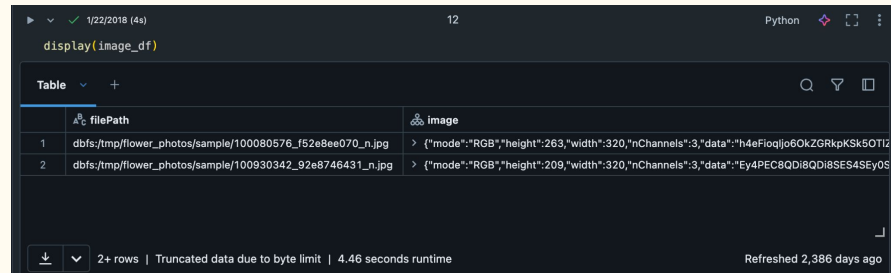


1/22/2018 (4s) 6

```
display(dbutils.fs.ls('file:/databricks/driver/flower_photos'))
```

	path	name	size
1	file:/databricks/driver/flower_photos/daisy/	daisy/	32768
2	file:/databricks/driver/flower_photos/dandelion/	dandelion/	49152
3	file:/databricks/driver/flower_photos/sunflowers/	sunflowers/	36864
4	file:/databricks/driver/flower_photos/LICENSE.txt	LICENSE.txt	418049
5	file:/databricks/driver/flower_photos/tulips/	tulips/	40960
6	file:/databricks/driver/flower_photos/roses/	roses/	36864

6 rows | 3.64 seconds runtime



1/22/2018 (4s) 12 Python

```
display(image_df)
```

	filePath	image
1	dbfs:/tmp/flower_photos/sample/100080576_152e8ee070_n.jpg	{ "mode": "RGB", "height": 263, "width": 320, "nChannels": 3, "data": "h4eFioqlc6OkZGRkpKSK50TIZ" }
2	dbfs:/tmp/flower_photos/sample/100930342_92e8746431_n.jpg	{ "mode": "RGB", "height": 209, "width": 320, "nChannels": 3, "data": "EY4PECBQDIBQDI8SES4SEyOS" }

2+ rows | Truncated data due to byte limit | 4.46 seconds runtime

Refreshed 2,386 days ago

Implementation - Transfer Learning

- **Step 5: Create Training and Test Data Frames**
 - Prepare and split data.
- **Step 6: Train and Evaluate Model**
 - Use feature extraction and logistic regression.
 - Evaluate model performance.

1/22/2018 (11m)

```
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)))
```

tested_df: pyspark.sql.dataframe.DataFrame

```
INFO:tensorflow:Froze 376 variables.
Converted 376 variables to const ops.
INFO:tensorflow:Froze 0 variables.
Converted 0 variables to const ops.
Test set accuracy = 0.971014492754
```

1/22/2018 (1s)

```
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([0.05, 0.05, 0.9]) # use large
non-community edition clusters)
daisy_train, daisy_test, _ = daisy_df.randomSplit([0.05, 0.05, 0.9]) # use large
non-community edition clusters)
train_df = tulips_train.unionAll(daisy_train)
test_df = tulips_test.unionAll(daisy_test)

# Under the hood, each of the partitions is fully loaded in memory, which may be expensive.
# This ensure that each of the partitions has a small size.
train_df = train_df.repartition(100)
test_df = test_df.repartition(100)
```

```
▶ tulips_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ daisy_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ tulips_train: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ tulips_test: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ daisy_train: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ daisy_test: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ _ : pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ train_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
▶ test_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
```

1/22/2018 (10m)

```
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=20, regParam=0.05, elasticNetParam=0.3, labelCol="label")
p = Pipeline(stages=[featurizer, lr])

p_model = p.fit(train_df)
```

Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.5/inception_v3_weights_tf_kernels_notop.h5

```
16384/87910968 [.....] - ETA: 0s
24576/87910968 [.....] - ETA: 7:06
57344/87910968 [.....] - ETA: 5:49
73728/87910968 [.....] - ETA: 6:31
106496/87910968 [.....] - ETA: 5:48
139264/87910968 [.....] - ETA: 5:27
196608/87910968 [.....] - ETA: 4:36
262144/87910968 [.....] - ETA: 4:01
352256/87910968 [.....] - ETA: 3:26
458752/87910968 [.....] - ETA: 2:59
```

Implementation - Applying Models at Scale

Step 7: Applying Deep Learning Models

- Use Transformers for TensorFlow and Keras models.
- Apply pre-trained models with **DeepImagePredictor**.

Step 7.1: Apply Popular Image Models

- Example: InceptionV3 model.

Step 7.2: Create Custom TensorFlow Graphs

- Use **TFImageTransformer**.

Step 7.3: Apply Keras Models

- Use **KerasImageFileTransformer**.

```
from keras.applications import InceptionV3

model = InceptionV3(weights="imagenet")
model.save('/tmp/model-full.h5') # saves to the local filesystem
# move to a permanent place for future use
dbfs_model_path = 'dbfs:/models/model-full.h5'
dbutils.fs.cp('file:/tmp/model-full.h5', dbfs_model_path)

Out[14]: True
```

```
from sparkdl import readImages, TFImageTransformer
from sparkdl.transformers import utils
import tensorflow as tf

image_df = readImages(sample_img_dir)

g = tf.Graph()
with g.as_default():
    image_arr = utils.imageInputPlaceholder()
    resized_images = tf.image.resize_images(image_arr, (299, 299))
    # the following step is not necessary for this graph, but can be for graphs with variables, etc
    frozen_graph = utils.stripAndFreezeGraph(g.as_graph_def(add_shapes=True), tf.Session(graph=g), (resized_images))

transformer = TFImageTransformer(inputCol="image", outputCol="transformed_image", graph=frozen_graph,
                                inputTensor=image_arr, outputTensor=resized_images,
                                outputMode="image")
tf_trans_df = transformer.transform(image_df)

image_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct]
tf_trans_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]

INFO:tensorflow:Froze 0 variables.
Converted 0 variables to const ops.
INFO:tensorflow:Froze 0 variables.
Converted 0 variables to const ops.
```

```
from sparkdl import readImages, DeepImagePredictor

image_df = readImages(sample_img_dir)

predictor = DeepImagePredictor(inputCol="image", outputCol="predicted_labels", modelName="InceptionV3",
                                decodePredictions=True, topK=10)
predictions_df = predictor.transform(image_df)

display(predictions_df.select("filePath", "predicted_labels"))

image_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct]
predictions_df: pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct ... 1 more field]
```

	filePath	predicted_labels
1	dbfs:/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg	[{"class": "n11939491", "description": "daisy", "probability": 0.8805}
2	dbfs:/tmp/flower_photos/sample/100930342_92e8746431_n.jpg	[{"class": "n03930313", "description": "picket_fence", "probability": 0.9538}
3	dbfs:/tmp/flower_photos/sample/10140303196_b88d3d6cec.jpg	[{"class": "n11939491", "description": "daisy", "probability": 0.9538}

Testing

Testing with Pre-trained Keras Model:

- Apply InceptionV3 to images in DataFrame.
- Preprocess images, generate predictions.

Results Table:

- **uri**: File paths of processed images.
- **predictions**: Model predictions with probabilities.

```
1/22/2018 (22s)

from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.image import img_to_array, load_img
import numpy as np
from pyspark.sql.types import StringType
from sparkdl import KerasImageFileTransformer

def loadAndPreprocessKerasInceptionV3(uri):
    # this is a typical way to load and prep images in keras
    image = img_to_array(load_img(uri, target_size=(299, 299))) # image dimensions for InceptionV3
    image = np.expand_dims(image, axis=0)
    return preprocess_input(image)

dbutils.fs.cp(dbfs_model_path, 'file:/tmp/model-full-tmp.h5')
transformer = KerasImageFileTransformer(inputCol="uri", outputCol="predictions",
                                       modelFile='file:/tmp/model-full-tmp.h5', # local file path for model
                                       imageLoader=loadAndPreprocessKerasInceptionV3,
                                       outputMode="vector")

files = ["/dbfs" + str(f.path)[5:] for f in dbutils.fs.ls(sample_img_dir)] # make "local" file paths for images
uri_df = sqlContext.createDataFrame(files, StringType()).toDF("uri")

keras_pred_df = transformer.transform(uri_df)

uri_df: pyspark.sql.dataframe.DataFrame = [uri: string]
keras_pred_df: pyspark.sql.dataframe.DataFrame

/databricks/python/local/lib/python2.7/site-packages/keras/models.py:255: UserWarning: No training configuration found in save file: the model was *not* compiled. Compile it manually.
warnings.warn('No training configuration found in save file: '
INFO:tensorflow:Froze 378 variables.
Converted 378 variables to const ops.
INFO:tensorflow:Froze 0 variables.
Converted 0 variables to const ops.
```

1/22/2018 (8s)

```
display(keras_pred_df.select("uri", "predictions"))
```

Table +

	A_c^B uri
1	/dbfs/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg
2	/dbfs/tmp/flower_photos/sample/100930342_92e8746431_n.jpg
3	/dbfs/tmp/flower_photos/sample/10140303196_b88d3d6cec.jpg

Enhancement Ideas

Future Improvements:

- Distributed Hyper-Parameter Tuning.
- SQL Functions for Deep Learning.
- Enhanced Image Processing.

Potential Applications:

- Real-time Image Classification.
- Large-scale Image Retrieval.
- Automated Feature Extraction.

Conclusion

Summary:

- Integrated DLP with Apache Spark for scalable applications.
- Efficiently processed large-scale image datasets.
- Implemented transfer learning and model evaluation.

Key Takeaways:

- Simplifies application of deep learning models.
- Provides tools for image processing and model deployment.
- Future enhancements will expand capabilities.

Reference

- https://hc.labnet.sfbu.edu/~henry/sfbu/course/machine_learning/deep_learning/slide/exercise_deep_learning.html
- <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bfcf/5669198905533692/3647723071348946/3983381308530741/latest.html>
- <https://github.com/databricks/spark-deep-learning>