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

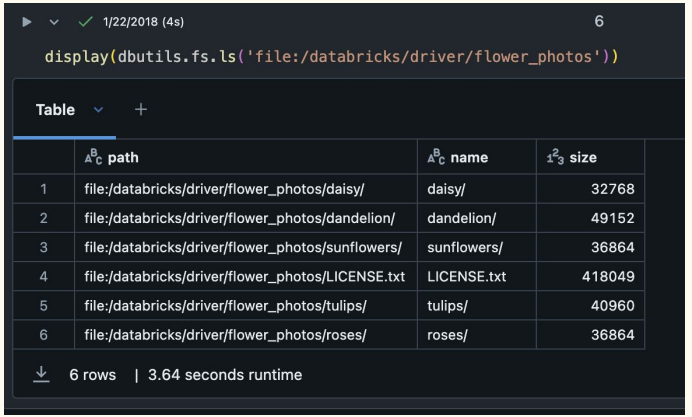
- Download and extract dataset.

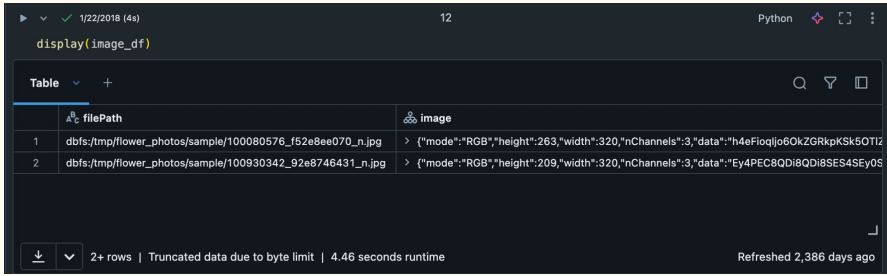
Step 3: Create Sample Set

- Generate smaller set for demonstration.

Step 4: Load Images into DataFrames

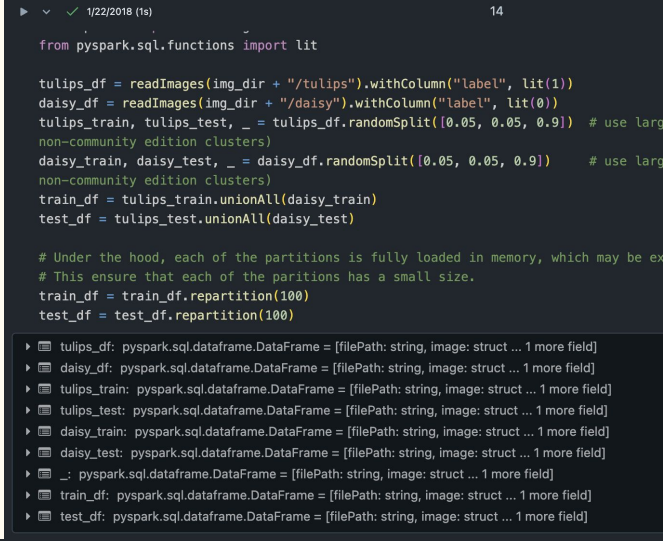
- Use DLP tools to read images into Spark DataFrames.

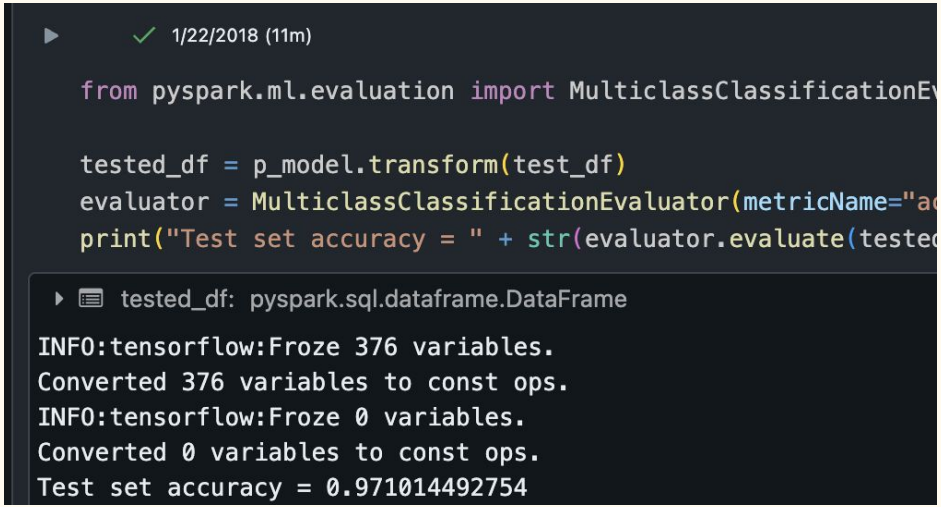


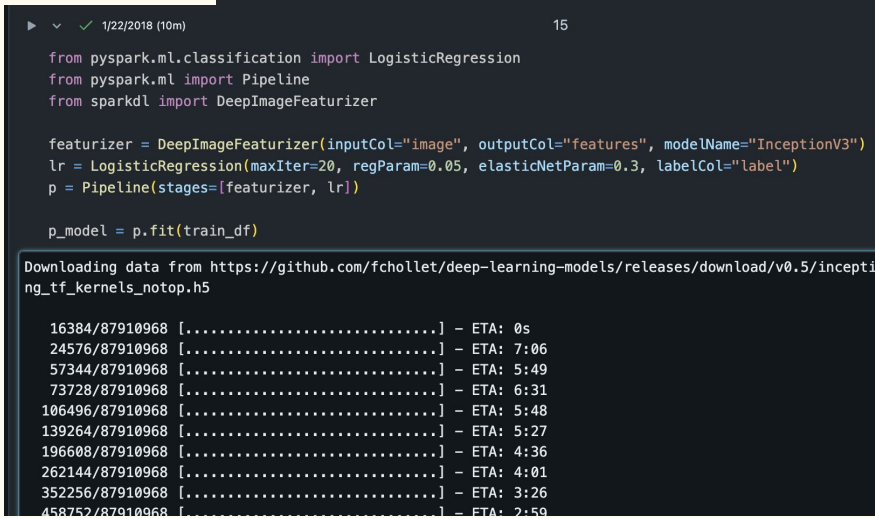


Implementation - Transfer Learning

- **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.







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

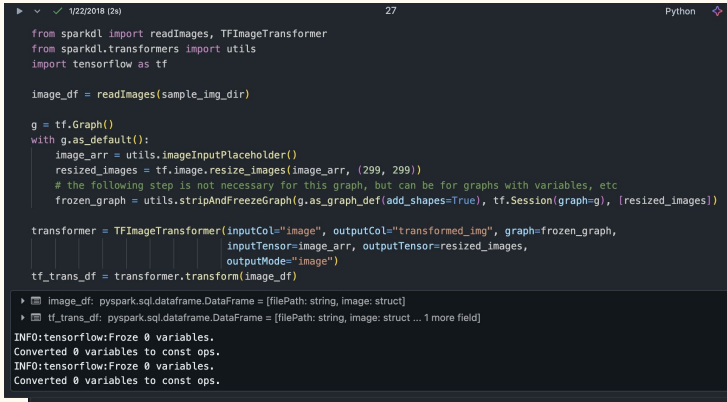
- Example: InceptionV3 model.

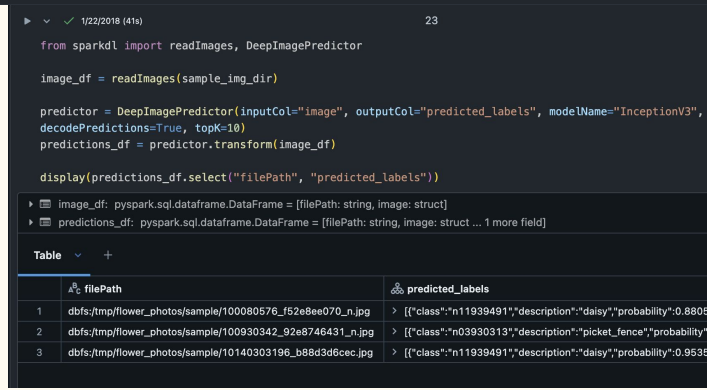
Step 7.2: Create Custom TensorFlow Graphs

- Use `TFImageTransformer`.

Step 7.3: Apply Keras Models

- Use `KerasImageFileTransformer`.





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) 32 Python

```
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='/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 +

	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/4027ec902e239c93eaaa8714f173bcfc/5669198905533692/3647723071348946/3983381308530741/latest.html
- https://github.com/databricks/spark-deep-learning