Deep Learning Pipelines for Apache Spark

CS570 - Big Data Processing and Analytics

Saron Haile 20069

Table of contents

01

Introduction

Overview

- Deep Learning Pipelines library by Databricks
- High-level APIs for scalable deep learning and transfer learning
- Integration with popular deep learning libraries (TensorFlow, Keras) and Spark MLlib

Objectives

- Demonstrate how to set up and use Deep Learning Pipelines
- Provide examples of TensorFlow and Keras integration
- Showcase practical applications and enhancements

Design

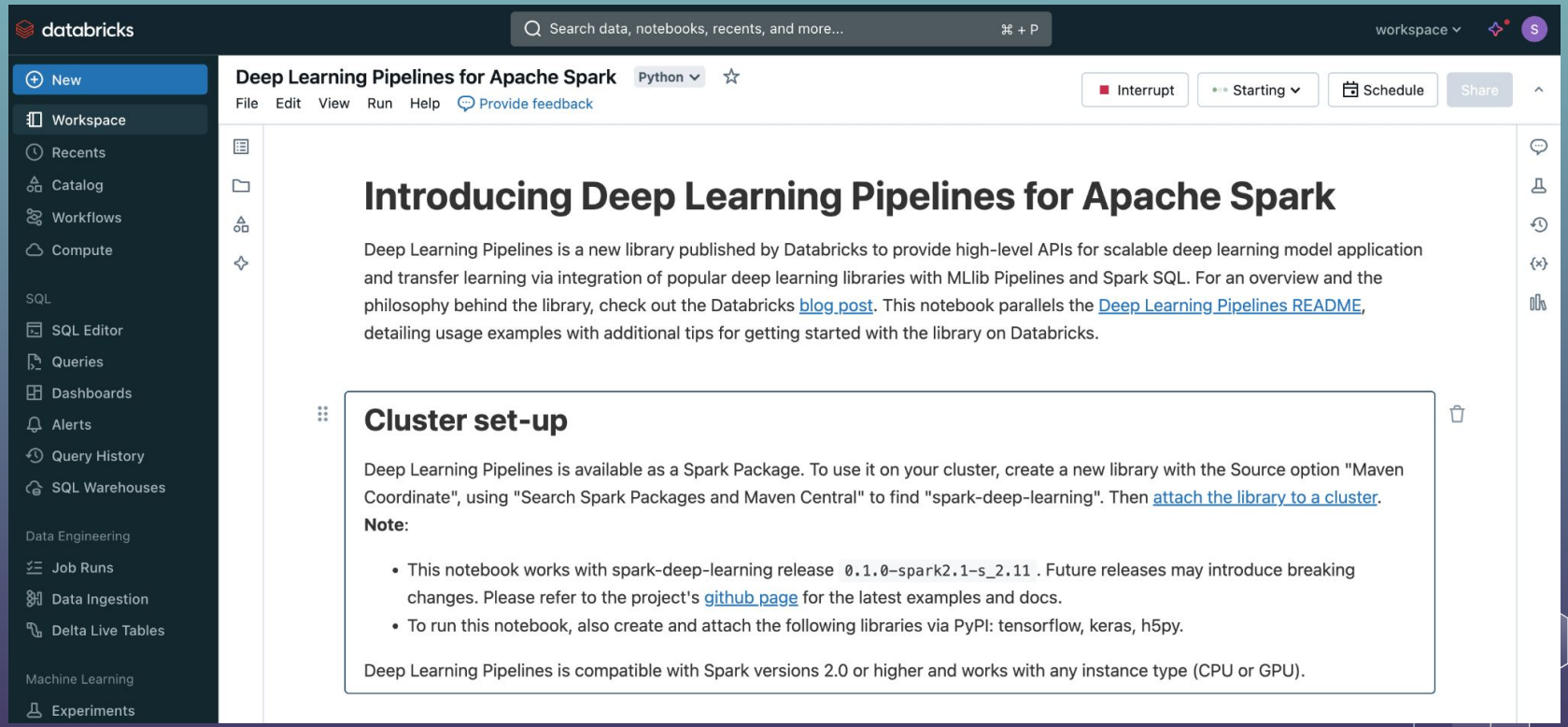
02

Cluster Setup

- **Installation:**
  - Add spark-deep-learning as a Maven coordinate library
  - Attach to the cluster
- **Compatibility:**
  - Spark versions 2.0 or higher
  - CPU or GPU instance types
- **Additional Libraries:**
  - TensorFlow, Keras, h5py via PyPI

Implementation and Testing

Step I: Run code on databricks



Step I: Run code on databricks

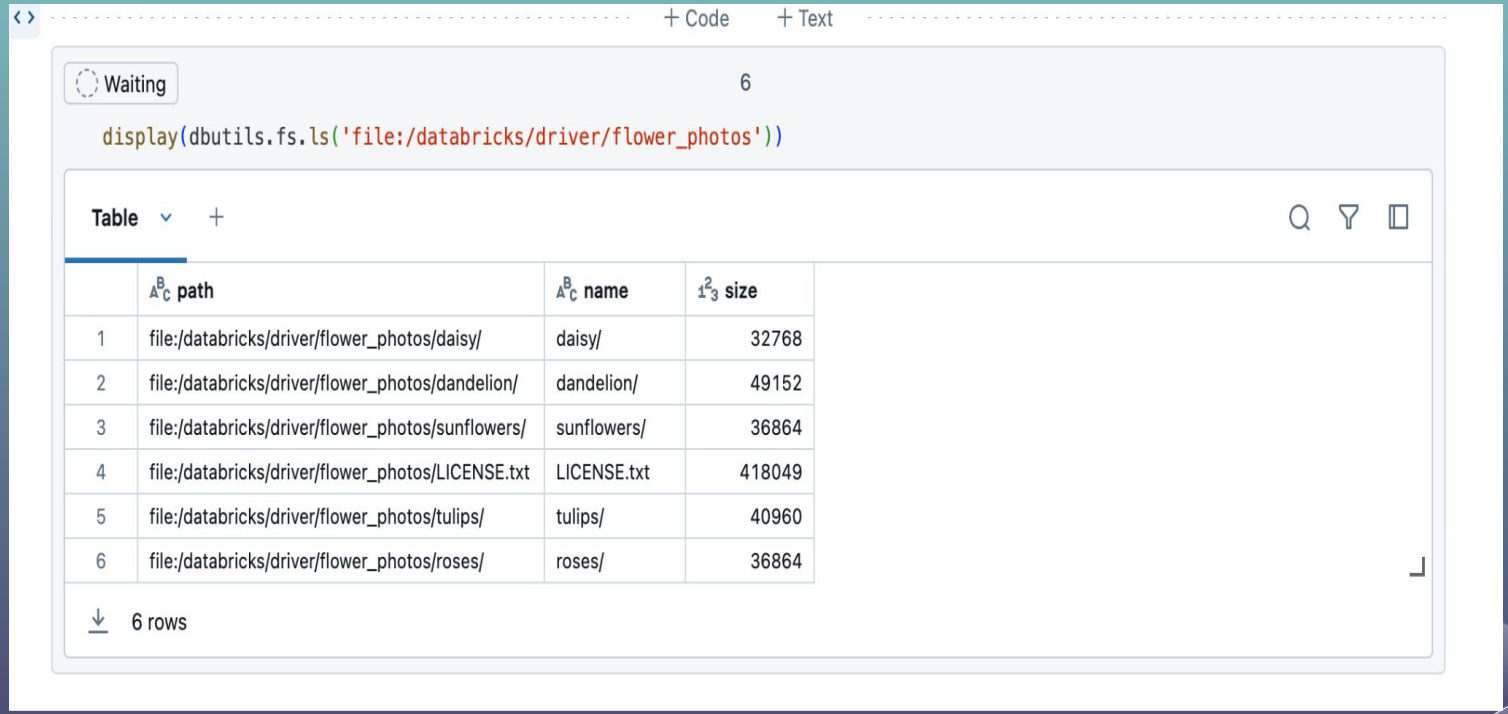
Let us first get some images to work with in this notebook. We'll use the flowers dataset from the TensorFlow retraining tutorial.

Waiting Starting Spark 5

```
%sh
curl -O http://download.tensorflow.org/example_images/flower_photos.tgz
tar xzf flower_photos.tgz
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
 12  218M   12 28.0M    0     0  24.0M      0  0:00:09  0:00:01  0:00:08 24.0M
100  218M  100  218M    0     0   114M      0  0:00:01  0:00:01 --:--:--  114M
tar: flower_photos/roses/14810868100_87eb739f26_m.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/1446090416_f0cad5fde4.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/15319767030_e6c5602a77_m.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/15032112248_30c5284e54_n.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/7211616670_2d49ecb3a5_m.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/15674450867_0ced942941_n.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/17158274118_00ec99a23c.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/14019883858_e5d2a0ec10_n.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/8035908422_87220425d2_n.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/14747962886_2bff6bb323_m.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/4356781875_92c5cd93c0.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/8524505546_b242bd4928_n.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/9406573080_60eab9278e_n.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
tar: flower_photos/roses/6039330368_c30ed224c4_m.jpg: Cannot change ownership to uid 270850, gid 5000: Invalid argument
```

Step 1: Run code on databricks



Step I: Run code on databricks

Waiting

7

```
# The 'file:/...' directory will be cleared out upon cluster termination. That doesn't matter for this example notebook,
but in most cases we'd want to store the images in a more permanent place. Let's move the files to dbfs so we can see
how to work with it in the use cases below.
img_dir = '/tmp/flower_photos'
dbutils.fs.mkdirs(img_dir)
dbutils.fs.cp('file:/databricks/driver/flower_photos/tulips', img_dir + "/tulips", recurse=True)
dbutils.fs.cp('file:/databricks/driver/flower_photos/daisy', img_dir + "/daisy", recurse=True)
dbutils.fs.cp('file:/databricks/driver/flower_photos/LICENSE.txt', img_dir)
display(dbutils.fs.ls(img_dir))
```

Table

	path	name	size
1	dbfs:/tmp/flower_photos/LICENSE.txt	LICENSE.txt	418049
2	dbfs:/tmp/flower_photos/daisy/	daisy/	0
3	dbfs:/tmp/flower_photos/tulips/	tulips/	0

3 rows

Step I: Run code on databricks

Waiting 8

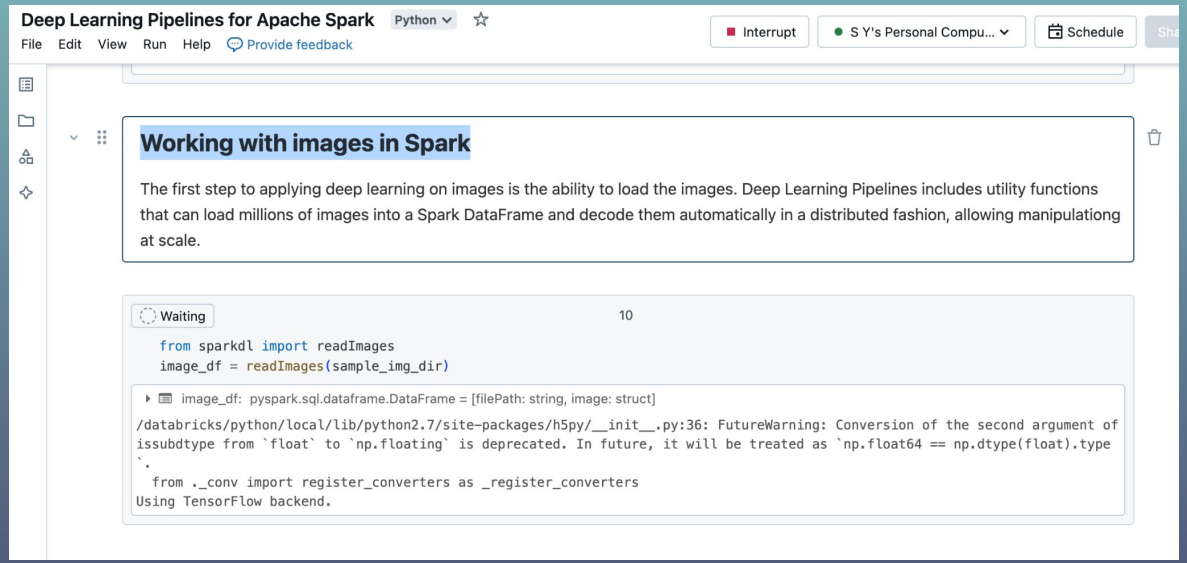
```
# Let's create a small sample set of images for quick demonstrations.
sample_img_dir = img_dir + "/sample"
dbutils.fs.mkdirs(sample_img_dir)
files = dbutils.fs.ls(img_dir + "/tulips")[0:1] + dbutils.fs.ls(img_dir + "/daisy")[0:2]
for f in files:
  dbutils.fs.cp(f.path, sample_img_dir)
display(dbutils.fs.ls(sample_img_dir))
```

Table

	path	name	size
1	dbfs:/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg	100080576_f52e8ee070_n.jpg	26797
2	dbfs:/tmp/flower_photos/sample/100930342_92e8746431_n.jpg	100930342_92e8746431_n.jpg	26200
3	dbfs:/tmp/flower_photos/sample/10140303196_b88d3d6cec.jpg	10140303196_b88d3d6cec.jpg	117247

3 rows

Step 2: Working with images in Spark



Step 2: Working with images in Spark

The resulting DataFrame contains a string column named "filePath" containing the path to each image file, and a image struct (" `SpImage` ") column called "image" containing the decoded image data.

Waiting

12

```
display(image_df)
```

Table

	filePath	image
1	dbfs:/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg	{"mode":"RGB","height":263,"width":320,"nChannels":3,"data":"h4eFioqljo6OkZGRkpKSk5
2	dbfs:/tmp/flower_photos/sample/100930342_92e8746431_n.jpg	{"mode":"RGB","height":209,"width":320,"nChannels":3,"data":"Ey4PEC8QDi8QDi8SES4SE

2+ rows | Truncated data due to byte limit

Step 3: Transfer Learning

Deep Learning Pipelines for Apache Spark Python

File Edit View Run Help Provide feedback

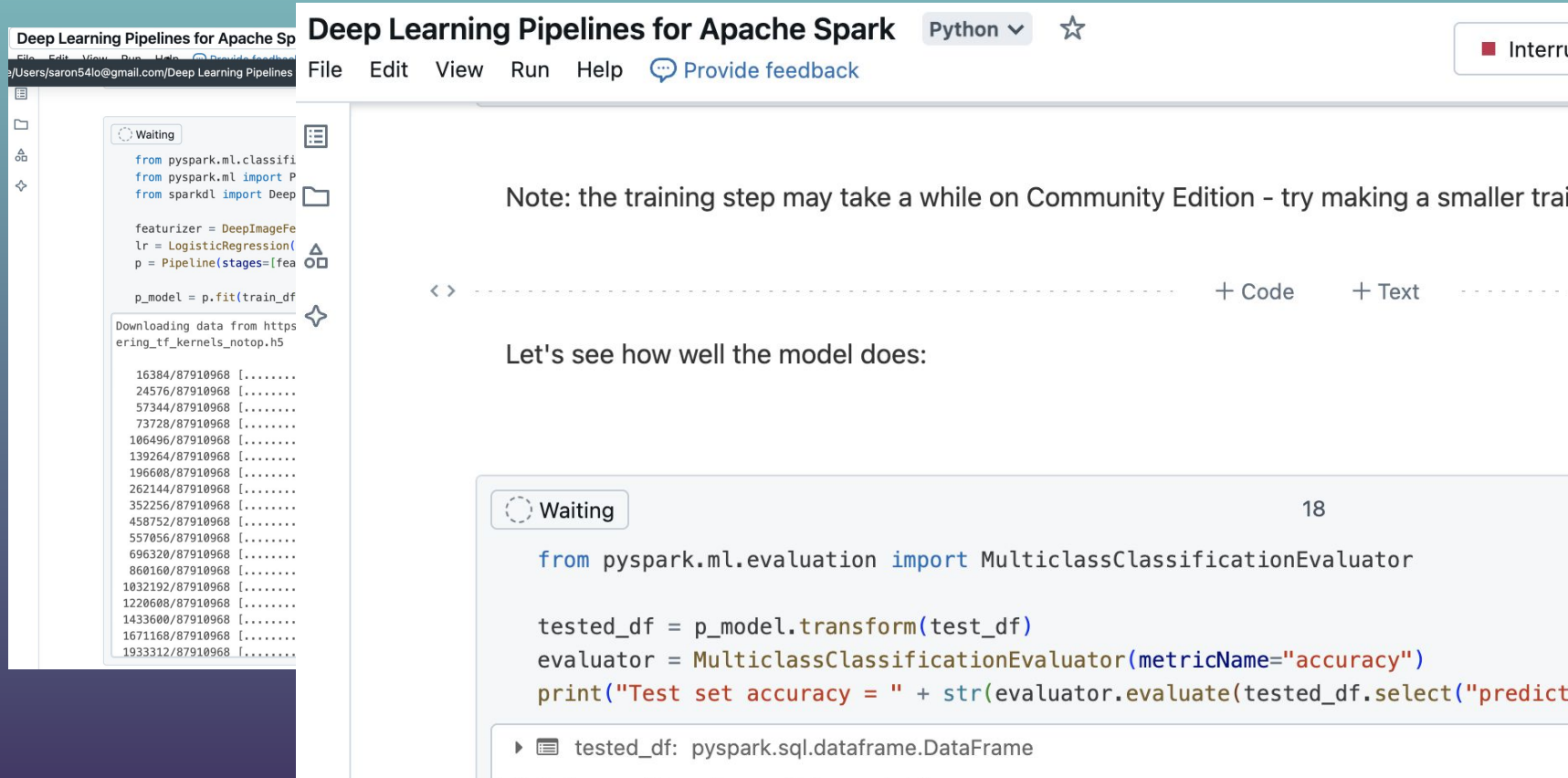
Interrupt | S Y's Personal Compu... | Schedule | Share

Waiting 14

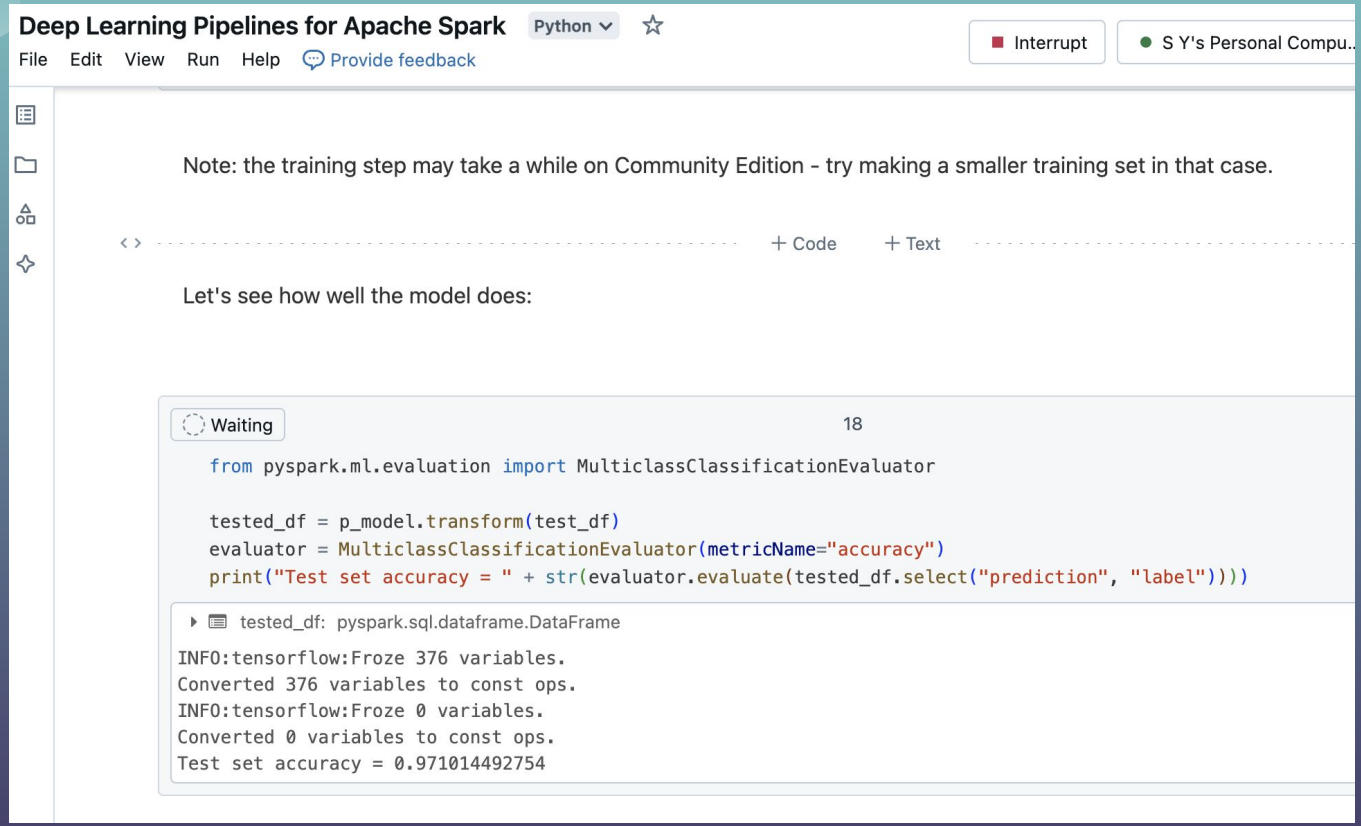
```
# Create training & test DataFrames for transfer learning - this piece of code is longer than transfer learning itself
below!
from sparkdl import readImages
from pyspark.sql.functions import lit

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([0.05, 0.05, 0.9])  # use larger training sets (e.g. [0.6, 0.4] for
non-community edition clusters)
daisy_train, daisy_test, _ = daisy_df.randomSplit([0.05, 0.05, 0.9])     # use larger training sets (e.g. [0.6, 0.4] for
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 paritions 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]

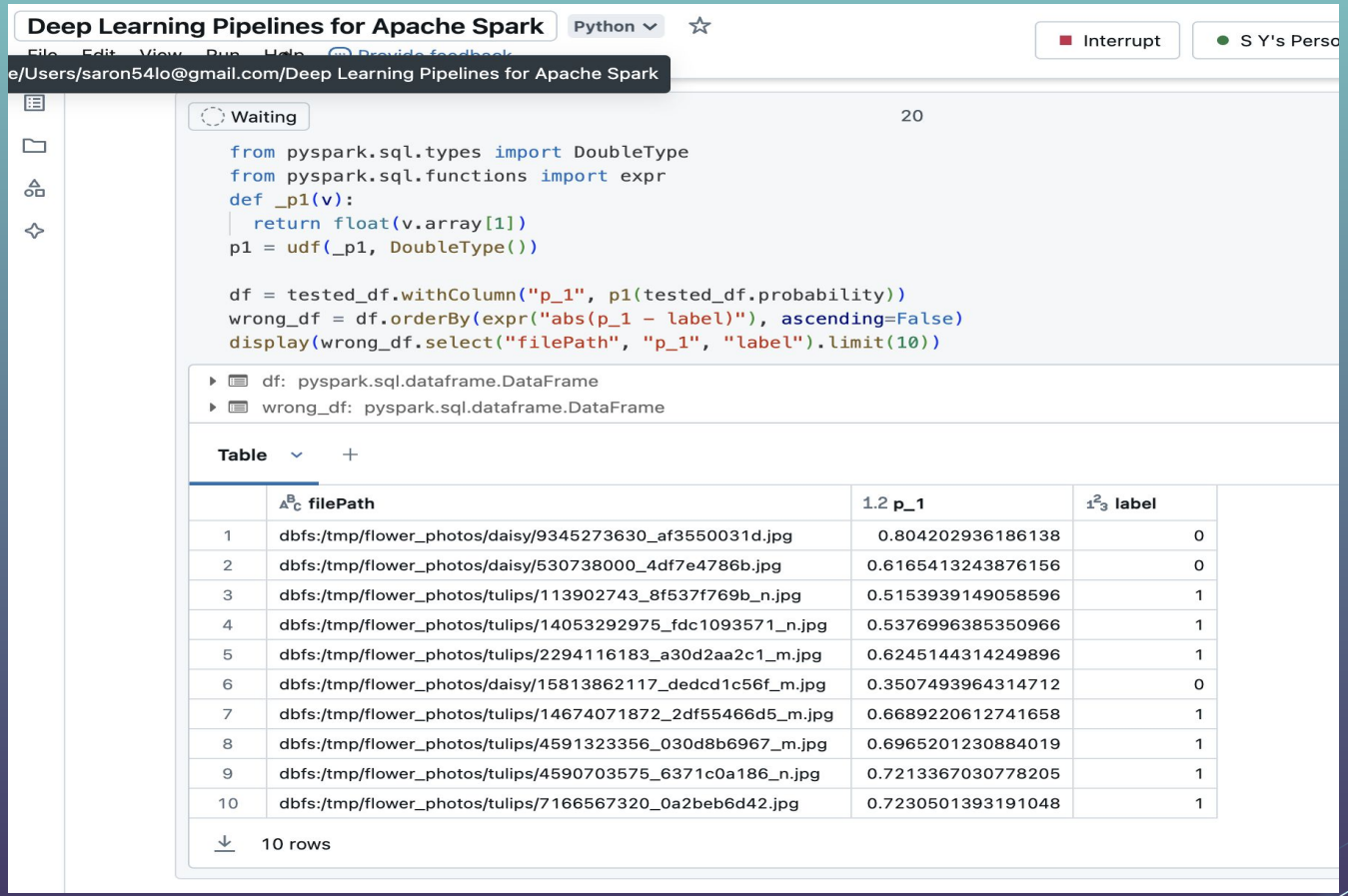
Step 3: Transfer Learning



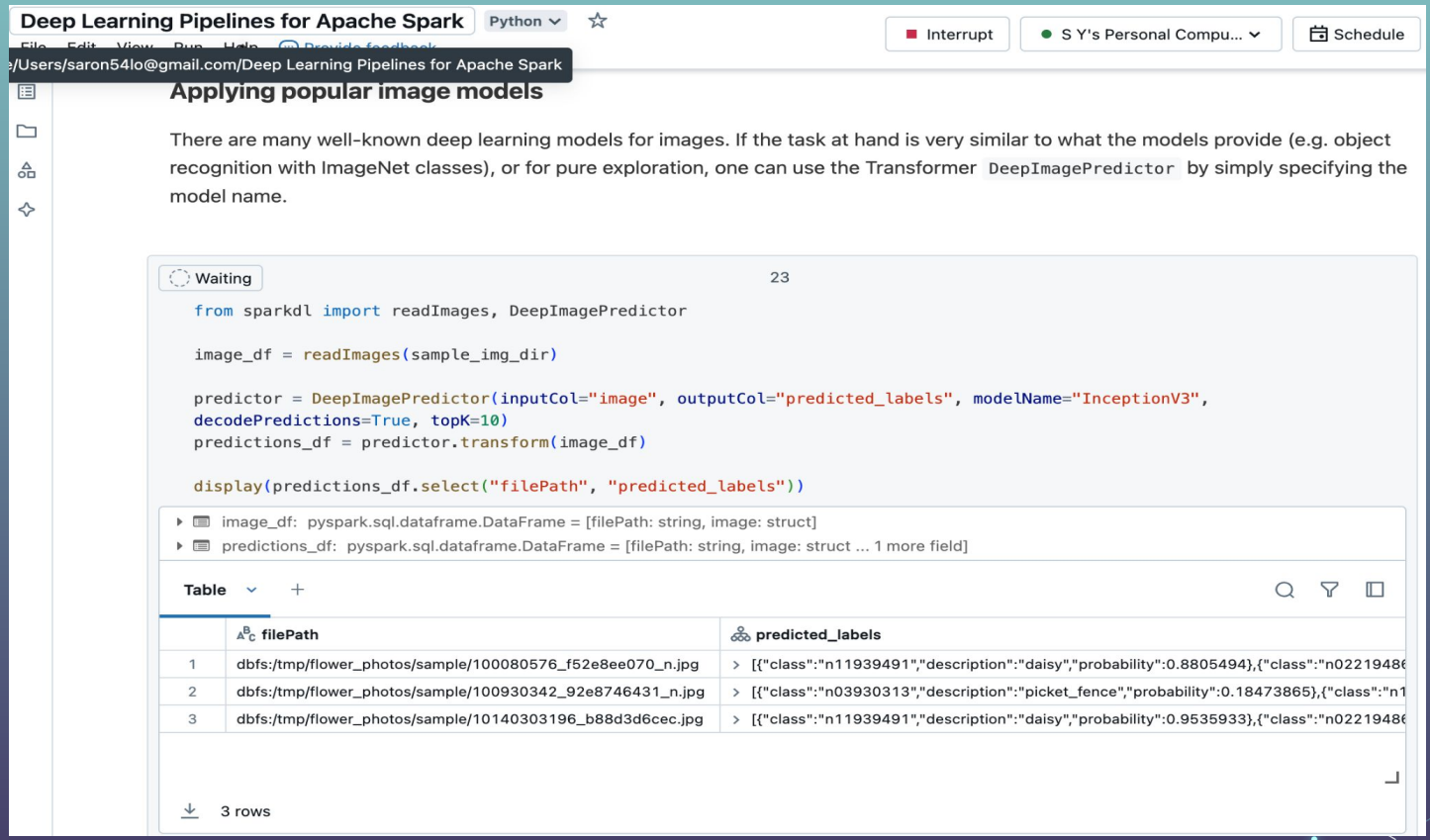
Step 3: Transfer Learning



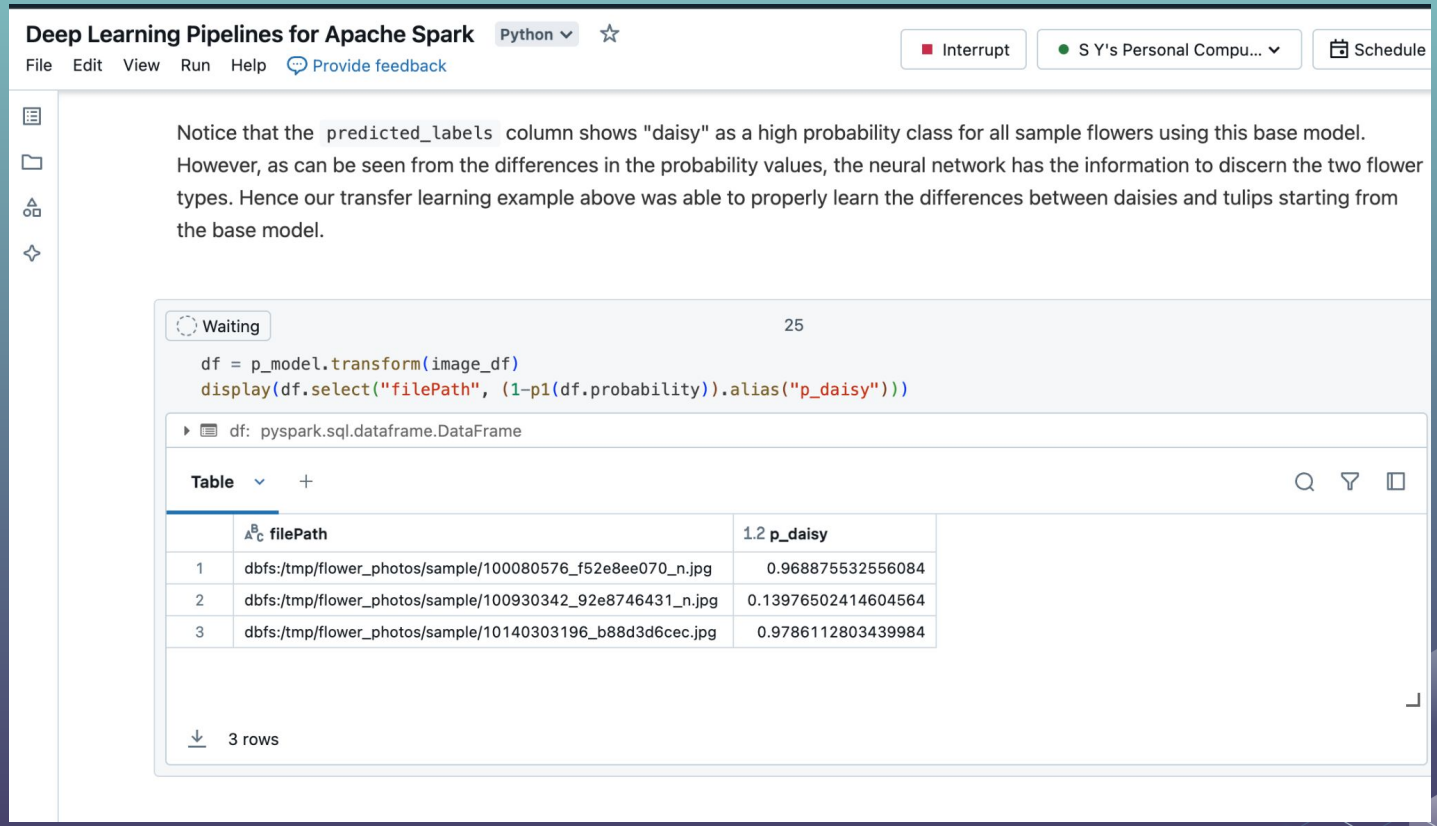
Step 3: Transfer Learning



Step 4: Applying Deep learning Models at scale



Step 4: Applying Deep learning Models at scale



Step 4: Applying Deep learning Models at scale

Deep Learning Pipelines for Apache Spark Python

File Edit View Run Help Provide feedback

Interrupt | S Y's Personal Compu... | Schedule

For TensorFlow users

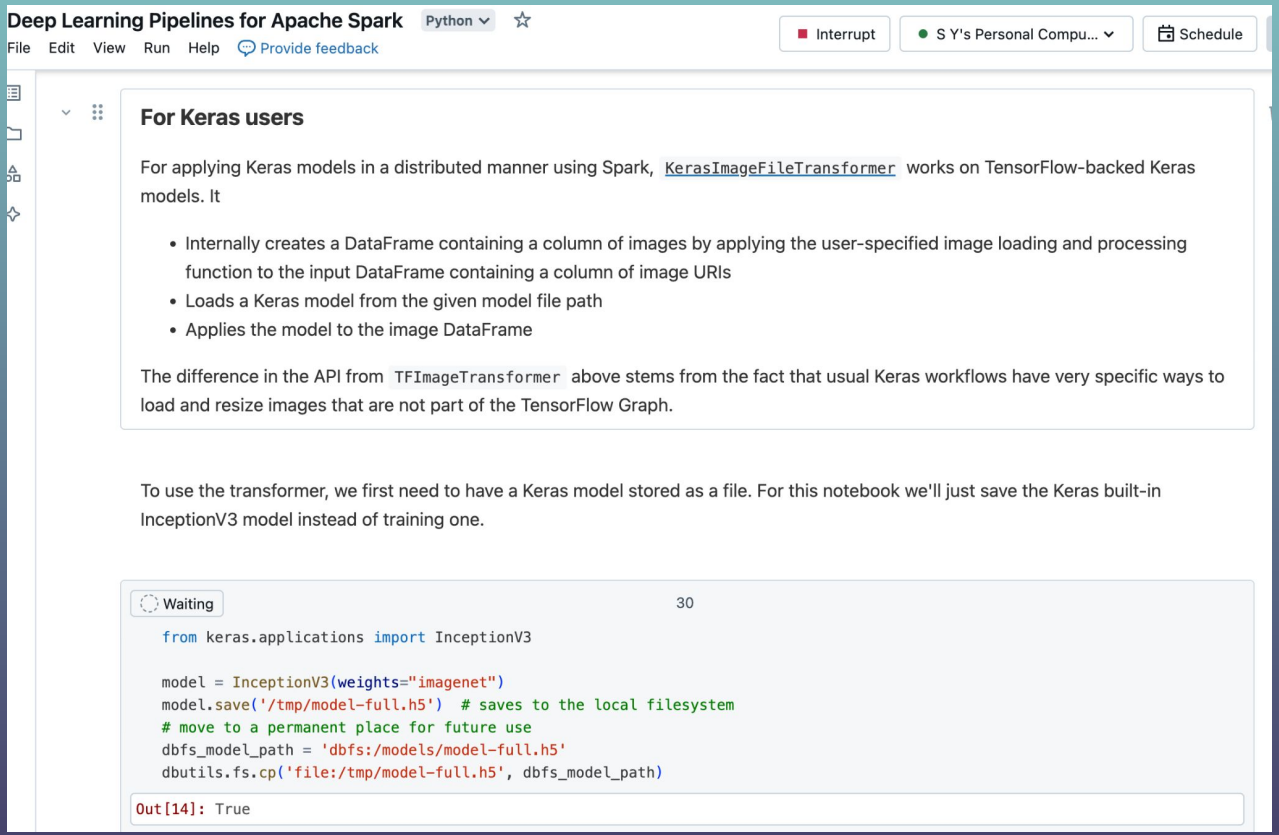
Deep Learning Pipelines provides a MLlib Transformer that will apply the given TensorFlow Graph to a DataFrame containing a column of images (e.g. loaded using the utilities described in the previous section). Here is a very simple example of how a TensorFlow Graph can be used with the Transformer. In practice, the TensorFlow Graph will likely be restored from files before calling `TFImageTransformer`.

Waiting 27

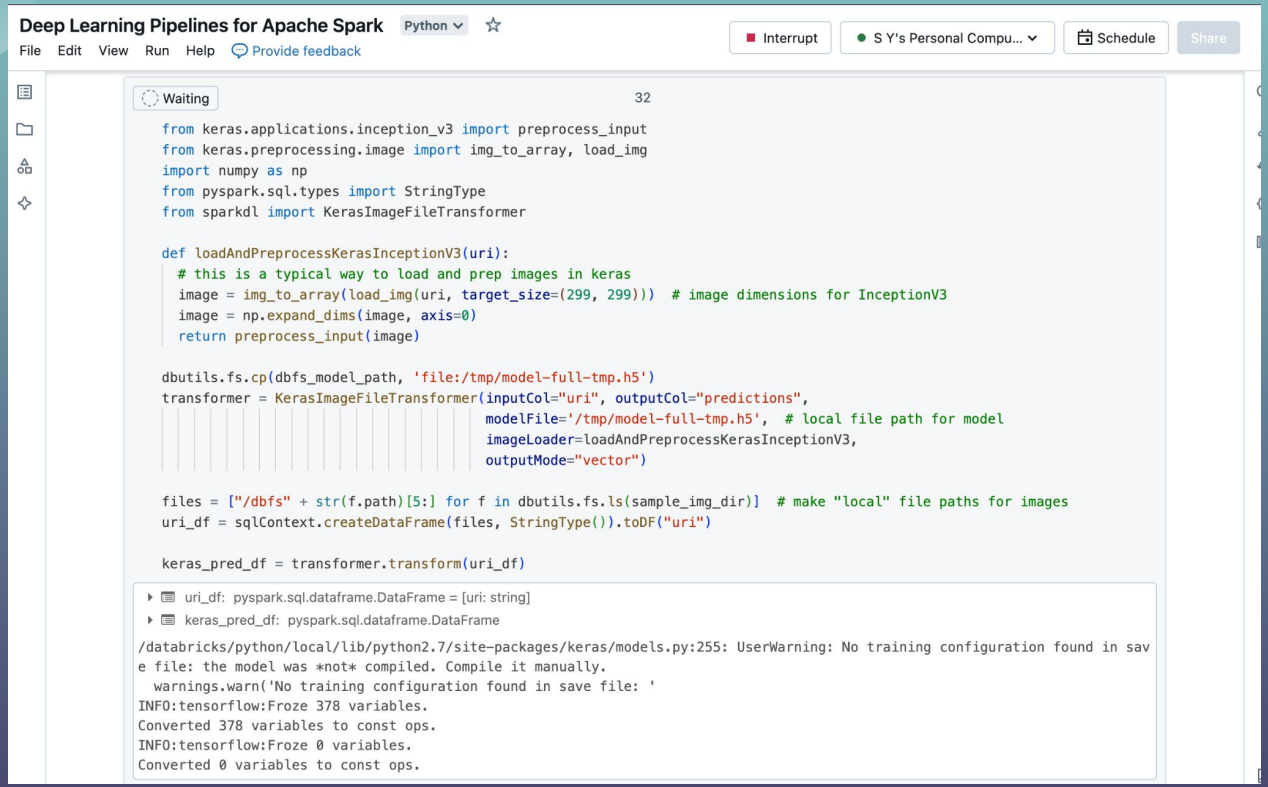
```
from sparkdl import readImages, TFImageTransformer
from sparkdl.transformers import utils
import tensorflow as tf

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_img", 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.
```

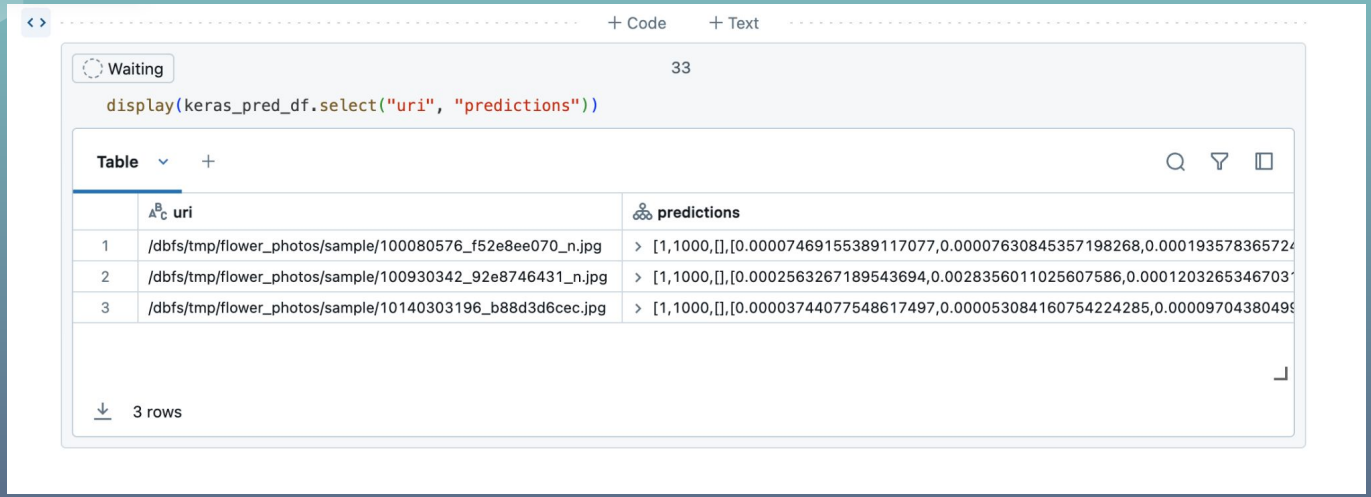
Step 4: Applying Deep learning Models at scale



Step 4: Applying Deep learning Models at scale



Step 4: Applying Deep learning Models at scale



Step 5: Cleanup Data Generated for this notebook

Clean up data generated for this notebook

Waiting

35

```
dbutils.fs.rm(img_dir, recurse=True)
dbutils.fs.rm(dbfs_model_path)
```

```
Out[17]: True
```

Enhancement 05

- SQL function deployment (upcoming feature)
- Distributed hyper-parameter tuning (upcoming feature)
- Integration with additional deep learning frameworks

Conclusion

06

Summary:

- Deep Learning Pipelines streamline deep learning model deployment on Spark
- Integration with TensorFlow and Keras provides flexibility
- Practical applications in image processing and beyond

Next Steps:

- Explore additional features and updates on the Deep Learning Pipelines GitHub page
- Stay updated with new releases and enhancements

GITHUB LINK

https://github.com/Sharon20222/Cloud-Computing/tree/main/Machine%20Learning/Apache%20Spark%20+%20Deep%20Learning

References

07

- https://github.com/databricks/spark-deep-learning
- https://docs.databricks.com/en/machine-learning/train-model/deep-learning.html
- https://www.databricks.com/blog/2017/06/06/databricks-vision-simplify-large-scale-deep-learning.html