

# Deep Learning Pipelines for Apache Spark on Databricks

Integrating Deep Learning with Spark for Scalable Applications

Niyat Habtom Seghid - 19967  
07/31/2024

# Contents:

1. Introduction
2. Design
3. Implementation
4. Test
5. Enhancement Ideas
6. Conclusion
7. Reference

Github Link:

<https://github.com/niyat33/Cloud-Computing/tree/a9337290725edd59bf68084a57c7eb932a56a332/Machine%20Learning/Apache%20Spark%20%2B%20Deep%20Learning>

# Introduction:

## Deep Learning Pipelines (DLP):

- A library by Databricks to simplify the application and transfer learning of deep learning models.
- Integrates popular deep learning libraries with Apache Spark's MLlib Pipelines and Spark SQL.
- Reference: Databricks blog post and the [Deep Learning Pipelines README](#).

## Objectives:

- Provide high-level APIs for scalable deep learning.
- Facilitate the application of deep learning models on large datasets.
- Enable transfer learning to adapt pre-trained models to new tasks with minimal effort.

## Project Aim:

- Develop a robust pipeline for applying deep learning models to large-scale image datasets.
- Implement transfer learning to fine-tune pre-trained models for specific tasks.
- Evaluate the scalability and performance of the pipeline using **Apache Spark**.

## Outcome:

- Create an efficient workflow for deep learning applications in image processing.
- Demonstrate the benefits of integrating deep learning with Spark for large datasets.



# Cont.

## Key Technologies Used:

- **Apache Spark:**
  - Used for distributed data processing and machine learning model training.
  - Provides scalability and efficiency for handling large datasets.
- **MLlib Pipelines:**
  - A component of Apache Spark for building and tuning machine learning models.
  - Supports the creation of complex workflows combining multiple stages of data processing and model training.
- **Spark SQL:**
  - Enables querying structured data using SQL within Spark.
  - Integrates with DLP to allow deep learning models to be applied directly within SQL queries.
- **Deep Learning Libraries (TensorFlow, Keras, etc.):**
  - TensorFlow and Keras are used for defining and training deep learning models.
  - DLP leverages these libraries to enable transfer learning and feature extraction in a Spark environment.



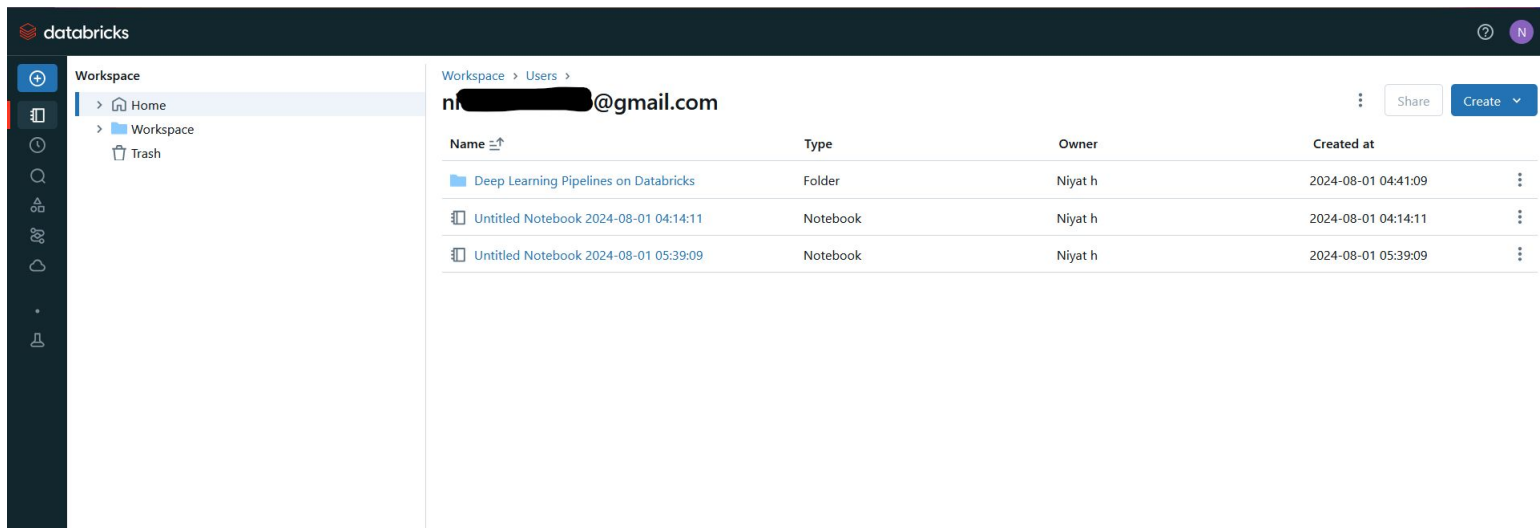
# Design

Deep Learning Pipelines provides a suite of tools around working with and processing images using deep learning. The tools can be categorized as

- **Image Data Handling:**
  - Tools for working with images natively in Spark DataFrames.
  - Support for reading, processing, and transforming image data at scale.
- **Transfer Learning:**
  - Provides mechanisms to adapt pre-trained deep learning models to new tasks.
  - Allows users to leverage existing models for feature extraction and fine-tuning.
- **Model Application at Scale:**
  - APIs for applying deep learning models to large datasets for predictions or feature transformations.
  - Integration with Spark's distributed processing capabilities ensures efficient execution.
- **Deploying Models as SQL Functions:**
  - Upcoming feature to allow deep learning models to be used directly within SQL queries.
  - Empowers non-programmers to utilize deep learning models through SQL.
- **Distributed Hyper-Parameter Tuning:**
  - Planned feature for optimizing model parameters using Spark MLlib Pipelines.
  - Aims to automate the tuning process to achieve optimal model performance.

# Implementation

- Create an account in databricks and create a workspace and a new notebook.



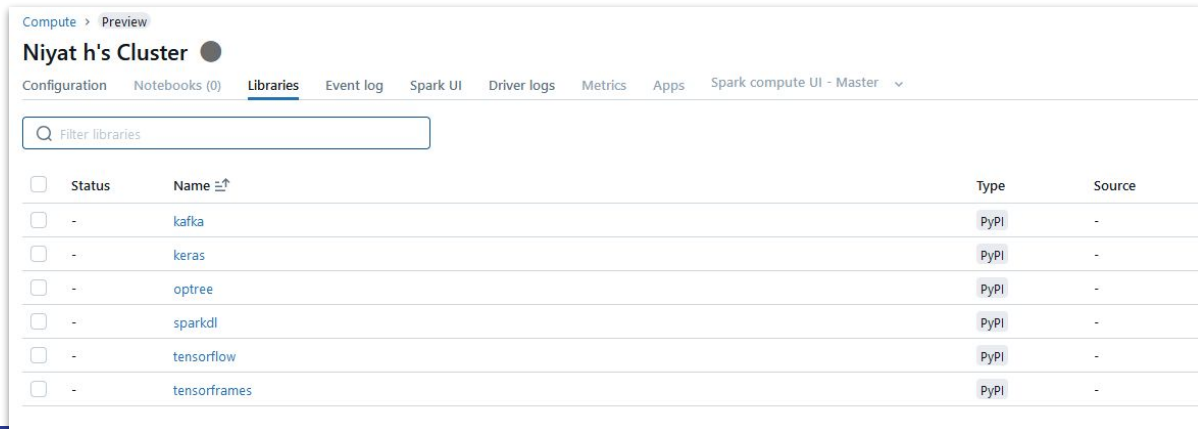
The screenshot displays the Databricks web interface. On the left, a sidebar shows the navigation menu with options like Home, Workspace, and Trash. The main area shows the 'Workspace' view for a user named 'ni[REDACTED]@gmail.com'. A table lists the contents of the workspace:

Name	Type	Owner	Created at
Deep Learning Pipelines on Databricks	Folder	Niyat h	2024-08-01 04:41:09
Untitled Notebook 2024-08-01 04:14:11	Notebook	Niyat h	2024-08-01 04:14:11
Untitled Notebook 2024-08-01 05:39:09	Notebook	Niyat h	2024-08-01 05:39:09

# Implementation: Image Data Handling

## Step 1: Setting Up the Cluster Environment

- **Install Libraries:** Deep Learning Pipelines is available as a Spark Package. To use it on your cluster, create a new library with the Source option "Maven Coordinate", using "Search Spark Packages and Maven Central" to find "spark-deep-learning" and then [attach the library to a cluster](#).
- **Install Dependencies:**
  - Add the Maven coordinate for **spark-deep-learning** to your Spark setup.
  - Install required Python libraries: TensorFlow, Keras, h5py, and spark-deep-learning.



Compute > Preview

**Niyat h's Cluster**

Configuration Notebooks (0) **Libraries** Event log Spark UI Driver logs Metrics Apps Spark compute UI - Master

Q Filter libraries

<input type="checkbox"/>	Status	Name	Type	Source
<input type="checkbox"/>	-	kafka	PyPI	-
<input type="checkbox"/>	-	keras	PyPI	-
<input type="checkbox"/>	-	optree	PyPI	-
<input type="checkbox"/>	-	sparkdl	PyPI	-
<input type="checkbox"/>	-	tensorflow	PyPI	-
<input type="checkbox"/>	-	tensorframes	PyPI	-

# Implementation: Image Data Handling

## Step 2: Obtain Image Dataset

- **Download and Extract Dataset:** We will use lowers dataset from the TensorFlow retraining tutorial here.

```
5 minutes ago (9s) 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
100	218M	100	218M	0	0	105M	0	0:00:02 0:00:02 --:--:-- 105M

06:34 AM (15s) 6

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

▶ (3) Spark Jobs

Table ▾ +

	path	name	size	modificationTime
1	file:/databricks/driver/flower_photos/daisy/	daisy/	36864	1455137572000
2	file:/databricks/driver/flower_photos/sunflowers/	sunflowers/	36864	1455137572000
3	file:/databricks/driver/flower_photos/roses/	roses/	36864	1455137572000
4	file:/databricks/driver/flower_photos/dandelion/	dandelion/	53248	1455137572000
5	file:/databricks/driver/flower_photos/LICENSE.txt	LICENSE.txt	418049	1454986755000
6	file:/databricks/driver/flower_photos/tulips/	tulips/	45056	1455137572000

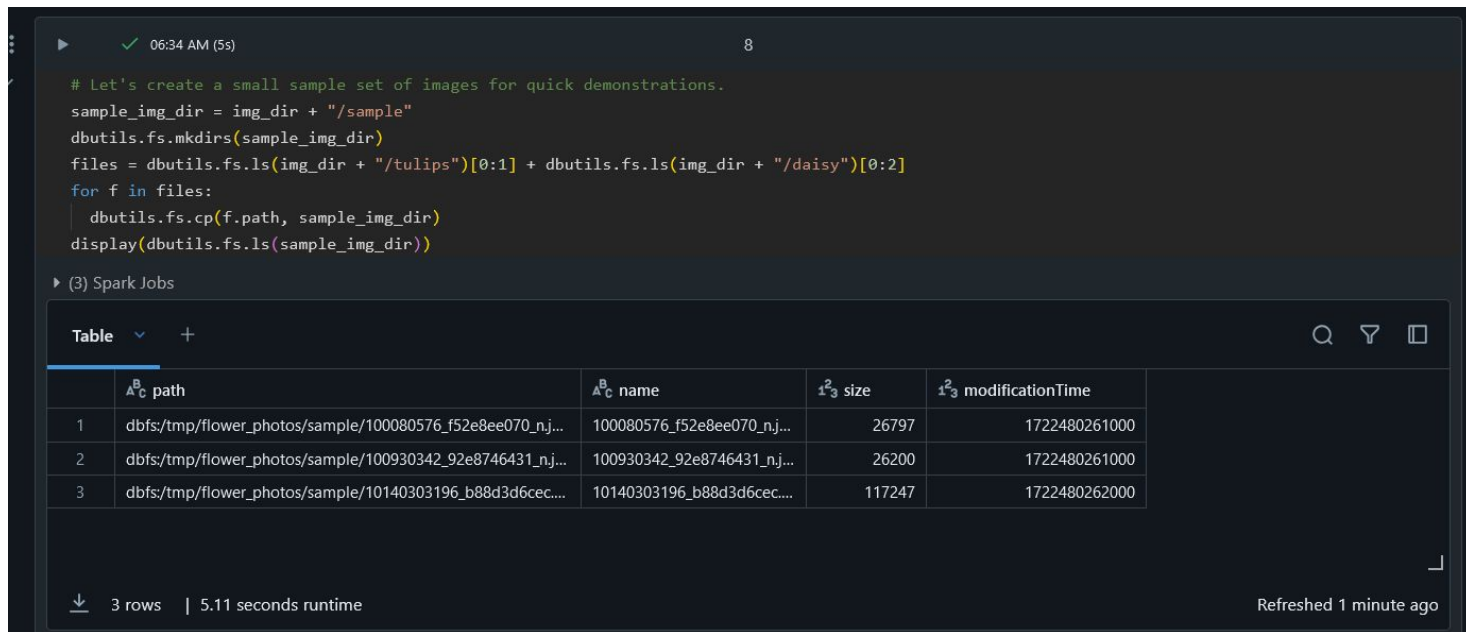
6 rows | 14.72 seconds runtime



# Implementation: Image Data Handling

## Step 3: Create a Sample Set of Images

- Create a smaller sample set of images for quick demonstrations.



The screenshot shows a Jupyter Notebook interface. The top part contains a code cell with the following Python code:

```
# 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))
```

Below the code cell, there is a table view showing the results of the file operations. The table has 5 columns: an index, the path, the name, the size, and the modification time. It contains 3 rows of data.

	path	name	size	modificationTime
1	dbfs/tmp/flower_photos/sample/100080576_f52e8ee070_nj...	100080576_f52e8ee070_nj...	26797	1722480261000
2	dbfs/tmp/flower_photos/sample/100930342_92e8746431_nj...	100930342_92e8746431_nj...	26200	1722480261000
3	dbfs/tmp/flower_photos/sample/10140303196_b88d3d6cec....	10140303196_b88d3d6cec....	117247	1722480262000

At the bottom of the table view, it indicates "3 rows" and "5.11 seconds runtime". A "Refreshed 1 minute ago" status is also shown in the bottom right corner.

# Implementation: Image Data Handling

## Step 4: Load Images into Spark DataFrames

- **Read Images:** Use DLP's tools to read images directly into Spark DataFrames.
- 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.

```
from sparkdl import readImages
image_df = readImages(sample_img_dir)
```

image\_df: pyspark.sql.dataframe.DataFrame

```
  filePath: string
  image: struct
    mode: string
    height: integer
    width: integer
    nChannels: integer
    data: binary
```

07:04 AM (25s) 12

display(image\_df)

(3) Spark Jobs

	filePath	image
1	dbfs/tmp/flower_photos/sample/100080576_f52e8ee070_nj...	> ("mode":"RGB","height":263,"width":320,"nChannels":3,"data":"h4eFioqljo6OKZGRkpKSK5OT1ZWxl5eZmZmZl5eVlZWl...
2	dbfs/tmp/flower_photos/sample/100930342_92e8746431_nj...	> ("mode":"RGB","height":209,"width":320,"nChannels":3,"data":"Ey4PEc8QDI8QDI8SE54SEy0SDyORD54RCIYPCCQNBmYm...
3	dbfs/tmp/flower_photos/sample/10140303196_b88d3d6cec...	> ("mode":"RGB","height":313,"width":500,"nChannels":3,"data":"1dzk1NTcz8/XztXbztfcz9LZ2dfi2t3m1trj1djh39/n19bc3...

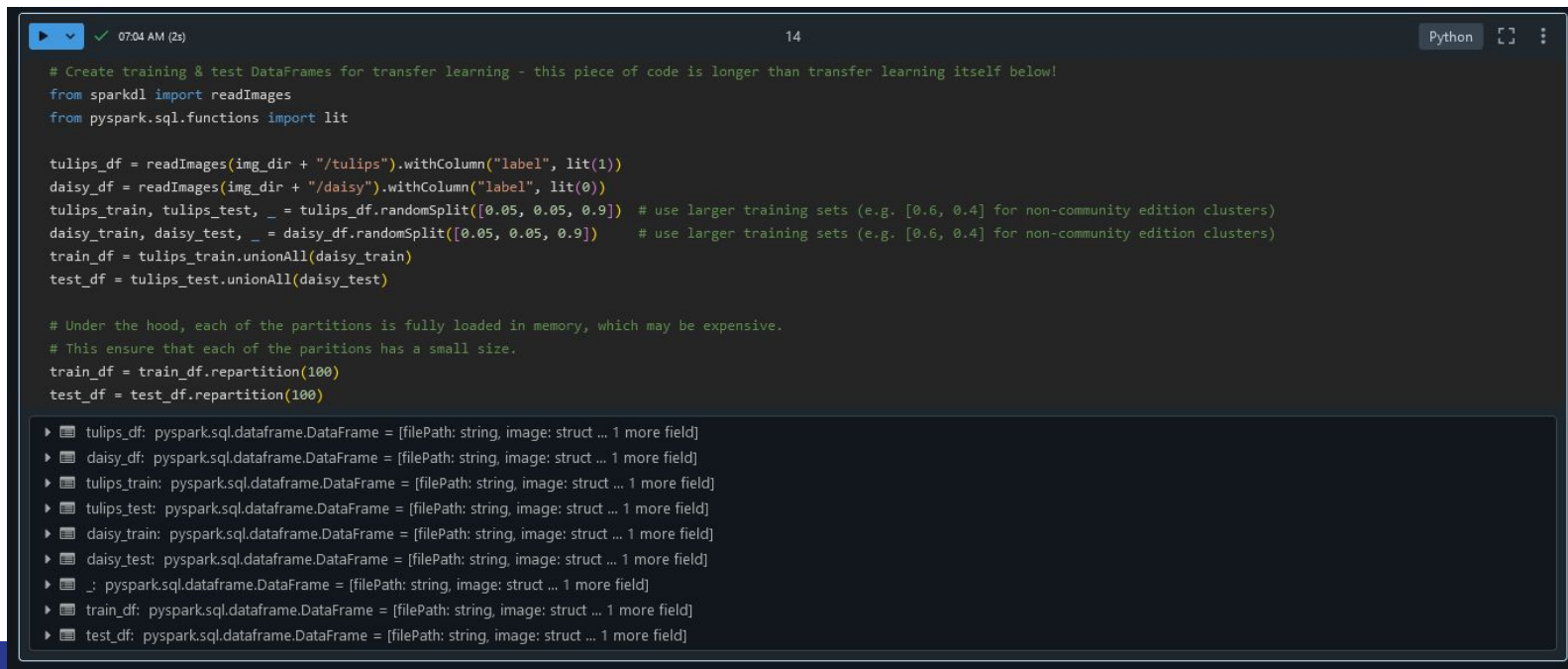
3 rows | 25.09 seconds runtime

Refreshed 1 minute ago

# Implementation: Transfer learning

## Step 5: Create Training and Test Data Frames for Transfer Learning

- **Data Preparation:**
  - Read images, assign labels, and split the data into training and test sets.



```
# 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 partitions has a small size.
train_df = train_df.repartition(100)
test_df = test_df.repartition(100)
```

Variable inspection panel:

- ▶ 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]

# Implementation: Transfer learning

## Step 6: Train and Evaluate the Model

- **Model Training and Evaluation:**
  - Train the model using a pipeline that includes feature extraction with a pre-trained model and logistic regression, then evaluate its performance

```
16
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_dim_ordering_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
557056/87910968 [.....] - ETA: 2:45
```



```
19
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"))))

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
```

# Implementation: Applying Deep Learning Models at Scale

## Step 7: Applying Deep Learning Models at Scale

- Spark DataFrames are a natural construct for applying deep learning models to large-scale datasets. Deep Learning Pipelines provides a set of (Spark MLlib) 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 workers.

### Step 7.1: Apply Popular Image Models

- Use the DeepImagePredictor transformer to apply popular pre-trained models for image classification.



# Implementation: Applying Deep Learning Models at Scale

```
24
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"))
```

	filePath	predicted_labels
1	dbfs/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg	> [{"class":"n11939491","description":"daisy","probability":0.8805494}, {"class":"n02219486","description":"ant","probability":0.0020712544...}
2	dbfs/tmp/flower_photos/sample/100930342_92e8746431_n.jpg	> [{"class":"n03930313","description":"picket_fence","probability":0.18473865}, {"class":"n11939491","description":"daisy","probability":0.1...}
3	dbfs/tmp/flower_photos/sample/10140303196_b88d3d6cec.j...	> [{"class":"n11939491","description":"daisy","probability":0.9533933}, {"class":"n02219486","description":"ant","probability":0.0005922901...}

3 rows

```
df = p_model.transform(image_df)
display(df.select("filePath", (1-p1(df.probability)).alias("p_daisy")))
```

	filePath	1.2 p_daisy
1	dbfs/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg	0.968875532556084
2	dbfs/tmp/flower_photos/sample/100930342_92e8746431_n.jpg	0.139765024146045...
3	dbfs/tmp/flower_photos/sample/10140303196_b88d3d6cec.j...	0.9786112803439984

- The `predicted\_labels` column shows that "daisy" is a high-probability class for all the sample flowers using the base model. However, the neural network can distinguish between the flower types based on the differences in the probability values. This demonstrates that our transfer learning example was able to learn and recognize the differences between daisies and tulips effectively, starting from the base model.

# Implementation: Applying Deep Learning Models at Scale

## Step 7.2: Create Custom TensorFlow Graphs - (For TensorFlow users)

- The `TFImageTransformer` allows users to apply custom TensorFlow Graphs to the image DataFrame. This is useful for specialized tasks where pre-trained models may not suffice.

```
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)
```

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

## Step 7.3: Apply Keras Models (For Keras users)

- The `KerasImageFileTransformer` allows users to apply Keras models to the DataFrame containing image URIs. This is useful for leveraging custom-trained Keras models within the Spark ecosystem.

```
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
```

# Testing:

## Step 8: Testing with Pre-trained Keras Model:

- This code snippet illustrates how to apply a pre-trained Keras model (InceptionV3) to images stored in a DataFrame.
- Images are loaded and preprocessed according to the input requirements of the InceptionV3 model.
- The pre-trained model is applied to the preprocessed images to generate predictions.
- The use of Spark DataFrames and transformers ensures that the process can scale to handle large datasets efficiently.

```
33

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)

/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.
```



# Testing:

This table displays the results of applying the pre-trained Keras model to the images in the sample directory using the **KerasImageFileTransformer**. Here's an explanation of what each column and the data within them represent:

- **uri**: This column lists the file paths (URIs) of the images that were processed.
- **predictions**: This column contains the model's predictions for each image. Each prediction includes a list of probabilities corresponding to different classes.

34

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

	uri	predictions
1	/dbfs/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg	> [1,1000,[],[0.00007469155389117077,0.00007630845357198268,0.0001935783657245338,0.00012283110118005425,0.00007464548980351537...
2	/dbfs/tmp/flower_photos/sample/100930342_92e8746431_n.jpg	> [1,1000,[],[0.0002563267189543694,0.0028356011025607586,0.00012032653467031196,0.00015315019118133932,0.00018672144506126642,...
3	/dbfs/tmp/flower_photos/sample/10140303196_b88d3d6cec.j...	> [1,1000,[],[0.00003744077548617497,0.000053084160754224285,0.00009704380499897525,0.00006655883771600202,0.000039256421587197...

3 rows

# Enhancement Ideas



## Future Improvements:

- **Distributed Hyper-Parameter Tuning:**
  - Implement support for automated hyper-parameter tuning using Spark MLlib Pipelines.
  - Optimize model parameters to achieve better performance.
- **SQL Functions for Deep Learning:**
  - Develop SQL functions that allow deep learning models to be applied directly within SQL queries.
  - Enable non-programmers to leverage deep learning capabilities through SQL.
- **Enhanced Image Processing:**
  - Improve tools for reading, processing, and augmenting image data.
  - Support more pre-trained models for diverse applications.

## Potential Applications:

- **Real-time Image Classification:**
  - Deploy models for real-time image classification tasks in applications such as surveillance and quality control.
- **Large-scale Image Retrieval:**
  - Use DLP for building large-scale image retrieval systems for search engines and recommendation systems.
- **Automated Feature Extraction:**
  - Apply DLP for automated feature extraction in various domains, such as medical imaging and satellite imagery.

# Conclusion



- **In this project we have:**
  - Successfully integrated Deep Learning Pipelines with Apache Spark, providing a scalable solution for deep learning applications.
  - Efficiently loaded, processed, and transformed large-scale image datasets using Spark DataFrames.
  - Implemented transfer learning with pre-trained models like InceptionV3, enabling feature extraction and fine-tuning for new tasks.
  - Trained and evaluated logistic regression models using Spark MLlib Pipelines, achieving high accuracy on test datasets.
  - Demonstrated the scalability and performance benefits of using Spark for distributed deep learning workflows.
  - Created a streamlined workflow for applying deep learning models to large datasets, from data ingestion to model evaluation.
- **Key Takeaways:**
  - DLP simplifies the application of deep learning models on large datasets.
  - Provides powerful tools for image processing, transfer learning, and model deployment.
  - Future enhancements will further expand the capabilities and usability of DLP.

# References:

[https://hc.labnet.sfbu.edu/~henry/sfbu/course/machine\\_learning/deep\\_learning/slide/exercise\\_deep\\_learning.html](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>

