# Deep Learning Pipelines on Databricks

databricks

CS570 Big Data Processing Project
By Feven Araya
Instructor: Dr. Chang, Henry

# Table of contents

# 01

# Introduction

- Deep Learning Pipelines is a new library from Databricks that offers high-level APIs to seamlessly integrate deep learning model application and transfer learning within Apache Spark's MLlib Pipelines and Spark SQL.
- This library leverages popular deep learning libraries to enable scalable deep learning model application.
- It allows users to work with images natively in Spark DataFrames, perform transfer learning, apply deep learning models at scale, and deploy models as SQL functions.

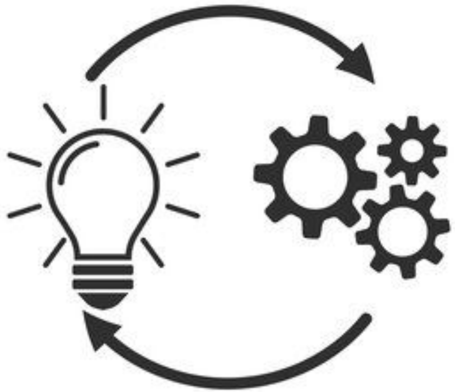
# 02
# Design

The design of Deep Learning Pipelines focuses on integrating deep learning capabilities within the Spark ecosystem. It provides tools for:

1. **Working with images in Spark DataFrames**: Utility functions load millions of images into a Spark DataFrame and decode them automatically in a distributed fashion.
2. **Transfer Learning**: Facilitates leveraging existing deep learning models with minimal code and run-time effort.
3. **Applying Deep Learning Models at Scale**: Uses Spark MLlib Transformers to apply TensorFlow Graphs and Keras Models efficiently.
4. **Future Enhancements**: Plans to include distributed hyper-parameter tuning via Spark MLlib Pipelines and deploying models as SQL functions.
    ○

# 03

# Implementation



## Cluster Setup

1. **Library Installation**: Deep Learning Pipelines is available as a Spark Package. Users need to create a new library with the Maven Coordinate source option to find "spark-deep-learning" and attach it to their cluster.
2. **Dependencies**: The notebook requires the following libraries via PyPI: `tensorflow`, `keras`, `h5py`.

1/22/2018 (4s) 6

```
%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
```

1/22/2018 (4s) 7

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

Table

| | 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 |

## Loading and Processing Images

1. **Loading Images**: The `readImages` function loads images into a Spark DataFrame.
2. **Creating Sample Set**: A smaller sample set is created for quick demonstrations by copying a subset of images.

```python
# The 'file:/...' directory will be cleared out upon cluster termination. That doesn't matter for
images in a more permanent place. Let's move the files to dbfs so we can see how to work with it
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 |

```python
# 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 |

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

```
image_df:  pyspark.sql.dataframe.DataFrame = [filePath: string, image: struct]
/databricks/python/local/lib/python2.7/site-packages/h5py/__init__.py:36: Fut
is deprecated. In future, it will be treated as `np.float64 == np.dtype(float
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
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":"h4eFioqljo6OkZGRkpKSk5OTlZWXl5eZmZmZl5eVlZWTlZW... |
| 2 | dbfs:/tmp/flower_photos/sample/100930342_92e8746431_n.jpg | {"mode":"RGB","height":209,"width":320,"nChannels":3,"data":"Ey4PEC8QDi8QDi8SES4SEy0SDy0RDS4RCiYPCCQNByMN... |

## Transfer Learning

1. **Data Preparation**: Images are labeled and split into training and test DataFrames.
2. **Model Training**: A logistic regression model is trained using features extracted by the `DeepImageFeaturizer` from the InceptionV3 model.
3. **Model Evaluation**: The trained model's accuracy is evaluated on the test set.

1/22/2018 (1s) 15 Python

```python
# 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)
```

```python
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)
```

```python
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"))))
```

```
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
```

```python
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import expr
def _p1(v):
  return float(v.array[1])
p1 = udf(_p1, DoubleType())

df = tested_df.withColumn("p_1", p1(tested_df.probability))
wrong_df = df.orderBy(expr("abs(p_1 - label)"), ascending=False)
display(wrong_df.select("filePath", "p_1", "label").limit(10))
```

df: pyspark.sql.dataframe.DataFrame

wrong_df: pyspark.sql.dataframe.DataFrame

Table

| | filePath | p_1 | label |
|---|---|---|---|
| 1 | dbfs:/tmp/flower_photos/daisy/9345273630_af3550031d.jpg | 0.804202936186138 | 0 |
| 2 | dbfs:/tmp/flower_photos/daisy/530738000_4df7e4786b.jpg | 0.6165413243876156 | 0 |
| 3 | dbfs:/tmp/flower_photos/tulips/113902743_8f537f769b_n.jpg | 0.5153939149058596 | 1 |

## Applying Deep Learning Models at Scale

1. **Using Pre-trained Models**: The `DeepImagePredictor` applies pre-trained models like InceptionV3 to images and returns predictions.
2. **Custom TensorFlow Graphs**: The `TFImageTransformer` applies a user-defined TensorFlow Graph to the image DataFrame.
3. **Keras Models**: The `KerasImageFileTransformer` loads a Keras model and applies it to the DataFrame containing image URIs.

```
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]

Table

| | 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... |
| 2 | dbfs:/tmp/flower_photos/sample/100930342_92e8746431_n.jpg | [{"class":"n03930313","description":"picket_fence","probability":0.18473865},{"class":"n11939491","description":"daisy","... |
| 3 | dbfs:/tmp/flower_photos/sample/10140303196_b88d3d6cec.jpg | [{"class":"n11939491","description":"daisy","probability":0.9535933},{"class":"n02219486","description":"ant","probability... |

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

df: pyspark.sql.dataframe.DataFrame

Table

| | filePath | p_daisy |
|---|---|---|
| 1 | dbfs:/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg | 0.968875532556084 |
| 2 | dbfs:/tmp/flower_photos/sample/100930342_92e8746431_n.jpg | 0.13976502414604564 |
| 3 | dbfs:/tmp/flower_photos/sample/10140303196_b88d3d6cec.jpg | 0.9786112803439984 |

```python
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.
```

```python
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)
```

```
ut[14]: True
```

# 04

# Test

**Model Accuracy**: The logistic regression model trained using transfer learning achieved an accuracy of 97.1% on the test set.
**Prediction Results**: Predictions from the pre-trained InceptionV3 model were displayed, showing high probabilities for relevant classes (e.g., "daisy").
**Error Analysis**: The DataFrame was ordered by the absolute difference between predicted probabilities and actual labels to identify misclassified images.

```
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. C
ile 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.
```

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

Table

| | uri | predictions |
|---|---|---|
| 1 | /dbfs/tmp/flower_photos/sample/100080576_f52e8ee070_n.jpg | [1,1000,[],[0.0000746915538911707 7,0.000076308453571982 68,0.0001935783657245338,0.00012283110118005 4... |
| 2 | /dbfs/tmp/flower_photos/sample/100930342_92e8746431_n.jpg | [1,1000,[],[0.0002563267189543694,0.002835601102560758 6,0.0001203265346703119 6,0.00015315019118133 93... |
| 3 | /dbfs/tmp/flower_photos/sample/10140303196_b88d3d6cec.jpg | [1,1000,[],[0.0000374407754861749 7,0.0000530841607542242 85,0.0000970438049989752 5,0.000066558837716 0... |

3 rows | 7.68 seconds runtime

Refreshed 2,3

1/22/2018 (23s)

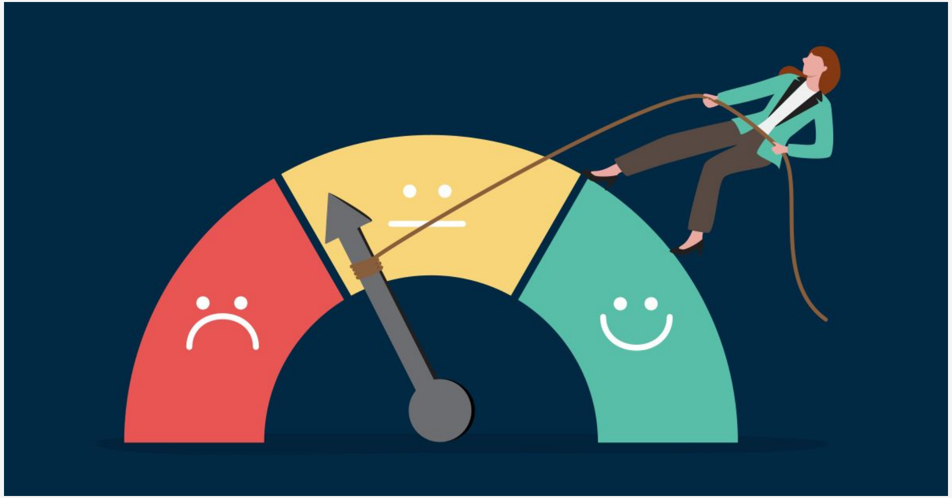
```
dbutils.fs.rm(img_dir, recurse=True)
dbutils.fs.rm(dbfs_model_path)
```

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

# 05

# Enhancements

Can we get better result?



## Enhancement

Future enhancements planned for Deep Learning Pipelines include:

1. **Distributed Hyper-parameter Tuning**: Leveraging Spark MLlib Pipelines to perform hyper-parameter tuning in a distributed manner.
2. **SQL Functions**: Deploying deep learning models as SQL functions to make them easily accessible via SQL queries.
3. **Enhanced Model Support**: Adding support for more models and improving the integration with other deep learning libraries.

# 06 Conclusion

- Deep Learning Pipelines offers a comprehensive suite of tools to integrate deep learning with Apache Spark, making it accessible for scalable data processing and model deployment.
- Its high-level APIs simplify the application of deep learning models and transfer learning, while future enhancements promise to further extend its capabilities, making it a valuable resource for data scientists and engineers working with big data and deep learning.

# 07

# References

Introducing Deep Learning Pipelines for Apache Spark

GitHub

# Thanks!

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

**Please keep this slide for attribution**