# Deep Learning using Apache Spark

By Shruti Kavishwar
San Francisco Bay University
Guided By: Prof. Henry Chang

# Agenda

- Introduction
- Data Preparation
- Model Architecture
- Training and Evaluation
- Result and Analysis
- Conclusion
- GitHub

# Introduction

## Introduction

This project explores the integration of Apache Spark with deep learning techniques to handle large-scale data and perform efficient model training and evaluation.

**Objective:** To leverage Apache Spark for scalable deep learning tasks.

# Data Preparation

- **Datasets:**
  - Tulips and daisies images, collected from specified directories.
- **Data Loading:**
  - Using TensorFlow to load images from directories.
  - Normalizing images and resizing to 224x224 pixels.
- **Data Split:**
  - Splitting the data into training (80%) and testing (20%) sets.
  - Ensuring stratified split based on labels to maintain class distribution.

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop2.7.tgz
!tar xf spark-3.1.2-bin-hadoop2.7.tgz
!pip install -q findspark
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop2.7"
```

```
import findspark
findspark.init('/content/spark-3.1.2-bin-hadoop2.7')
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

```
%%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  79.2M      0  0:00:02  0:00:02 --:--:-- 79.2M
```

```
[ ] # %% [code]
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
directory = 'flower_photos'
for filename in os.listdir(directory):
    print(filename)
```

```
tulips
dandelion
LICENSE.txt
daisy
roses
sunflowers
```

```
[ ] import os
    img_dir = '/content/flower_photos'
    os.makedirs(img_dir + "/tulips", exist_ok=True)
    os.makedirs(img_dir + "/daisy", exist_ok=True)
```

```python
import shutil
import os

source_dir = '/content/flower_photos'
img_dir = 'content/photos'  # Make sure this is defined correctly

def copy_tree(src, dst):
    try:
        if not os.path.exists(dst):
            os.makedirs(dst)
        shutil.copytree(src, dst, dirs_exist_ok=True)
        print(f"Successfully copied from {src} to {dst}")
    except Exception as e:
        print(f"Error copying from {src} to {dst}: {e}")

copy_tree(os.path.join(source_dir, 'tulips'), os.path.join(img_dir, 'tulips'))
copy_tree(os.path.join(source_dir, 'daisy'), os.path.join(img_dir, 'daisy'))

try:
    shutil.copy(os.path.join(source_dir, 'LICENSE.txt'), img_dir)
    print("Successfully copied LICENSE.txt")
except Exception as e:
    print(f"Error copying LICENSE.txt: {e}")
```

```
Successfully copied from /content/flower_photos/tulips to content/photos/tulips
Successfully copied from /content/flower_photos/daisy to content/photos/daisy
Successfully copied LICENSE.txt
```

```python
import pandas as pd
import tensorflow as tf
import os
from sklearn.model_selection import train_test_split
```

```python
# Function to load images from a directory and assign labels
def load_images_and_labels(directory, label):
    image_paths = [os.path.join(directory, fname) for fname in os.listdir(directory) if os.path.isfile(os.path.join(director
    images = []
    labels = []
    for path in image_paths:
        img = tf.io.read_file(path)
        img = tf.image.decode_image(img, channels=3)
        img = tf.image.resize(img, [224, 224])  # Resize to a standard size
        img = img / 255.0  # Normalize to [0, 1] range
        images.append(img.numpy())
        labels.append(label)
    return images, labels

# Load images and labels
tulips_images, tulips_labels = load_images_and_labels(tulips_dir, 1)
daisy_images, daisy_labels = load_images_and_labels(daisy_dir, 0)

# Combine images and labels into a DataFrame
df = pd.DataFrame({
    'image': tulips_images + daisy_images,
    'label': tulips_labels + daisy_labels
})

# Split data into training and testing sets
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['label'])

# Convert DataFrames to TensorFlow Datasets
def df_to_tf_dataset(df, batch_size=32):
    dataset = tf.data.Dataset.from_tensor_slices((list(df['image']), list(df['label'])))
    dataset = dataset.shuffle(buffer_size=len(df))
    dataset = dataset.batch(batch_size)
    return dataset

train_ds = df_to_tf_dataset(train_df)
test_ds = df_to_tf_dataset(test_df)
```

# Model Architecture

- **Model**: Logistic Regression
- **Key Features:**
    - Logistic regression is a linear model used for binary classification.
    - The model predicts the probability of a binary outcome (e.g., tulip or daisy).
- **Hyperparameters:**
    - Learning rate, regularization parameters.

```python
# Split data into training and testing sets
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['label'])

# Extract features from training and testing data
X_train = extract_features(train_df['image'].tolist())
y_train = train_df['label'].values
X_test = extract_features(test_df['image'].tolist())
y_test = test_df['label'].values

# Create and train the Logistic Regression model using the 'saga' solver
lr = LogisticRegression(max_iter=20, solver='saga', penalty='elasticnet', l1_ratio=0.3, C=1/0.05)

# Train the model
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Test set accuracy = {accuracy:.3f}")
```

# Training and Evaluation

- **Training Process:**
  - Distributed training using Apache Spark.
  - Leveraging Spark's DataFrame API for efficient data handling.
- **Evaluation Metrics:**
  - Accuracy: Measures the correctness of predictions.
  - Loss: Measures the error in predictions.
  - Confusion Matrix: Provides detailed insight into the classification performance.

```
# Split data into training and testing sets
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['label'])

# Extract features from training and testing data
X_train = extract_features(train_df['image'].tolist())
y_train = train_df['label'].values
X_test = extract_features(test_df['image'].tolist())
y_test = test_df['label'].values

# Create and train the Logistic Regression model using the 'saga' solver
lr = LogisticRegression(max_iter=20, solver='saga', penalty='elasticnet', l1_ratio=0.3, C=1/0.05)

# Train the model
lr.fit(X_train, y_train)

# Make predictions on the test set
```

# Result and Analysis

- **Performance Metrics:**
  - Accuracy: 0.819
- **Visualizations:**
  - Confusion Matrix: Show true positive, true negative, false positive, and false negative rates.
  - Sample Predictions: Examples of correctly and incorrectly classified images.

```
[ ] from sklearn.metrics import accuracy_score

    # Make predictions on the test set
    y_pred = lr.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Test set accuracy = {accuracy:.3f}")
```

```
Test set accuracy = 0.819
```

# Conclusion

- **Summary of Findings:**
  - Apache Spark effectively handles large-scale data and accelerates the training process.
  - The logistic regression model achieved satisfactory accuracy on the image classification task.
- **Future Work:**
  - Explore more complex models for improved performance.
  - Apply this approach to other datasets and domains.
  - Optimize Spark configurations for even better performance.

# GitHub Link

https://github.com/ShrutiK02/Cloud-Computing/tree/d56bc39a875d21e3e4fc4399939ac3aaf283bd53/Machine%20Learning/Apache%20Spark%20%2B%20Deep%20Learning