Deep Learning Pipelines for Apache Spark

Natnael Haile
ID: 20007
August 3, 2024

Table of Contents

Introduction

- **Overview of Deep Learning:**
  - Definition: A subset of machine learning that uses neural networks with many layers.
  - Applications: Image recognition, natural language processing, autonomous systems, etc.
- **Apache Spark:**
  - Definition: An open-source, distributed computing system for big data processing.
  - Capabilities: Data processing, in-memory computing, large-scale data analytics.
- **Purpose of Integration:**
  - Enhance Spark’s capabilities with deep learning functionalities.
  - Provide scalable and efficient data processing for training and inference.

Introduction: Objective

- **Challenges in Big Data Analytics:**
  - Handling large datasets efficiently.
  - Integration of various data processing and machine learning tasks.
- **Need for Deep Learning Pipelines:**
  - Streamline the process of training deep learning models on large datasets.
  - Improve model deployment and management within a Spark ecosystem.

Design: Objectives

Pipeline Architecture:

- **Data Ingestion:**
  - Methods for importing data into Spark (e.g., HDFS, S3).
  - Data preprocessing and transformation techniques.
- **Model Training:**
  - **Utilizing Spa**rk's distributed computing for model training.
  - Integration with deep learning frameworks (e.g., TensorFlow, PyTorch).
- **Model Deployment:**
  - Steps to deploy trained models for predictions.
  - Integration with Spark's streaming and batch processing capabilities.

Scalability and Efficiency:

- Techniques for optimizing performance and resource usage.
- Handling large-scale data and complex models efficiently.

Design: Key Components

- **Data Processing:**
  - Spark DataFrames and Datasets for handling large volumes of data.
  - Feature extraction and normalization techniques.
- **Deep Learning Frameworks:**
  - Overview of supported frameworks and their integration with Spark.
  - Benefits and challenges of using each framework.
- **Pipeline Management:**
  - Tools and libraries for managing and monitoring pipelines.
  - Best practices for maintaining pipeline efficiency and reliability.

Implementation: Environment Setup

- Open Google Colab and download Tensorflow, hadoop, and pyspark and start a spark session.

```
1m
!pip install pyspark tensorflow

Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 3.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
```

```
31s
[2] !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
```

```
0s
[3] 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"
```

```
8s
[4] import findspark
    findspark.init('/content/spark-3.1.2-bin-hadoop2.7')
    from pyspark.sql import SparkSession
    spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Implementation: Data Retrieval

- Get the flower dataset and make directories for it.

```
from google.colab import drive
drive.mount('/content/drive')
```

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

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

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

```
[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  96.5M      0  0:00:02  0:00:02 --:--:-- 96.5M
```

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

Implementation: Data Preparation

- Use shutil to load the images to apply deep learning on them. This is to Ensure that the dataset is properly organized and available at the correct path as it's crucial for smooth data processing and model training in Apache Spark.

```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
```

Implementation: Data Preparation

- Perform transfer learning on images by creating a sample subset of images from the `tulips` and `daisy` directories and copying them to a new directory (`sample_img_dir`) in order to prepare a small, manageable set of images from the larger dataset. This subset can be used for initial testing, debugging, or demonstration purposes..

```
img_dir = '/content/content/photos'  # Ensure this is correctly defined
sample_img_dir = os.path.join(img_dir, 'sample')

# Create the sample image directory
os.makedirs(sample_img_dir, exist_ok=True)

# List files in 'tulips' and 'daisy' directories
def list_files(directory, num_files):
    return sorted([os.path.join(directory, f) for f in os.listdir(directory) if os.path.isfile(os.path.join(directory, f))])[:num_files]

tulips_files = list_files(os.path.join(img_dir, 'tulips'), 1)
daisy_files = list_files(os.path.join(img_dir, 'daisy'), 2)
files = tulips_files + daisy_files

# Copy selected files to 'sample_img_dir'
for file_path in files:
    try:
        shutil.copy(file_path, sample_img_dir)
        print(f"Copied {file_path} to {sample_img_dir}")
    except Exception as e:
        print(f"Error copying {file_path}: {e}")

# List and display contents of the sample image directory
sample_img_dir_contents = os.listdir(sample_img_dir)
print("Contents of sample_img_dir:")
for item in sample_img_dir_contents:
    print(item)
```

```
Copied /content/content/photos/tulips/100930342_92e8746431_n.jpg to /content/content/photos/sample
Copied /content/content/photos/daisy/100080576_f52e8ee070_n.jpg to /content/content/photos/sample
Copied /content/content/photos/daisy/10140303196_b88d3d6cec.jpg to /content/content/photos/sample
Contents of sample_img_dir:
10140303196_b88d3d6cec.jpg
100930342_92e8746431_n.jpg
100080576_f52e8ee070_n.jpg
```

Implementation: Data Preparation

- Next load, preprocess, and display images from a specified directory using TensorFlow and Matplotlib before feeding the data to a deep learning model.
- Displaying a subset of images also helps in validating the preprocessing steps and verifying the quality of the input data.

```
# Define the directory containing the images
sample_img_dir = '/content/content/photos/sample'  # Update this path as needed

# Function to load and preprocess images
def load_and_preprocess_image(path):
    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
    return img

# Create a TensorFlow Dataset to read images
def load_images_from_directory(directory):
    image_paths = [os.path.join(directory, fname) for fname in os.listdir(directory) if os.path.isfile(os.path.join(directory, fname))]
    image_paths_ds = tf.data.Dataset.from_tensor_slices(image_paths)
    image_ds = image_paths_ds.map(lambda path: tf.numpy_function(func=lambda p: load_and_preprocess_image(p.decode('utf-8')), inp=[path], Tout=tf.float32))
    return image_ds

# Load images
image_ds = load_images_from_directory(sample_img_

# Display some images
def display_images(image_ds, num_images=3):
    plt.figure(figsize=(10, 10))
    for i, img in enumerate(image_ds.take(num_ima
        plt.subplot(1, num_images, i + 1)
        plt.imshow(img.numpy())
        plt.axis('off')
    plt.show()

# Display images
display_images(image_ds)
```

Implementation: Data Preparation

- Prepare image data by loading, preprocessing, labeling, and structuring it for training and testing a deep learning model.
- Ensure that the dataset is properly divided into training and testing subsets, which is crucial for evaluating model performance.
- Convert the data into TensorFlow Datasets, which are optimized for performance and scalability during model training.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Define directories
tulips_dir = '/content/content/photos/tulips'
daisy_dir = '/content/content/photos/daisy'

# 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 o
    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 st
        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)
```

Implementation: Feature Extraction & Classification

- Use a pre-trained model like InceptionV3 for feature extraction that leverages powerful learned representations from large-scale datasets.
- Apply Logistic Regression on these features as it provides a simple yet effective method for classification tasks.
- Accuracy is used to assess the performance of the Logistic Regression model, providing insight into how well the model distinguishes between tulips and daisies based on the extracted features.

```python
import numpy as np

# Define directories
tulips_dir = '/content/content/photos/tulips'
daisy_dir = '/content/content/photos/daisy'

# 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(directory, fname))]
    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, [299, 299])  # Resize to a standard size
        img = img / 255.0  # Normalize to [0, 1] range
        images.append(img.numpy())
        labels.append(label)
    return images, labels

# Extract features from images
def extract_features(images):
    # Load the InceptionV3 model pre-trained on ImageNet
    model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet', pooling='avg')

    # Preprocess images
    images = [tf.image.resize(img, [299, 299]) for img in images]
    images = np.array([tf.keras.applications.inception_v3.preprocess_input(img) for img in images])

    # Extract features
    features = model.predict(images)
    return features
```

```python
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 = 0.784
```

Test: Error Analysis & Model Improvement

- Analyze the most erroneous predictions made by the model.
- By examining the cases where the predicted probability deviates the most from the true label, you can gain insights into which images or conditions the model struggles with.
- Identify and understand these errors as they can help you in refining your model, improving data quality, or adjusting model parameters.
- This information can be used in visualization or reporting to demonstrate areas where the model needs improvement.

```python
import pandas as pd
import numpy as np

# Assuming df is the DataFrame with 'probability' and 'label' columns
# For demonstration purposes, I'll create a sample DataFrame
# Replace this with actual DataFrame creation/loading code
data = {
    'filePath': ['path/to/img1', 'path/to/img2', 'path/to/img3'],
    'probability': [np.array([0.1, 0.9]), np.array([0.8, 0.2]), np.array([0.4, 0.6])],
    'label': [1, 0, 1]
}
df = pd.DataFrame(data)

# Extract the probability for the positive class (index 1)
df['p_1'] = df['probability'].apply(lambda v: float(v[1]))

# Sort DataFrame by the absolute difference between 'p_1' and 'label'
df['abs_diff'] = np.abs(df['p_1'] - df['label'])
wrong_df = df.sort_values(by='abs_diff', ascending=False)

# Display the top 10 rows
print(wrong_df[['filePath', 'p_1', 'label']].head(10))
```

```
       filePath  p_1  label
2  path/to/img3  0.6      1
1  path/to/img2  0.2      0
0  path/to/img1  0.9      1
```

Test: Model Utilization and Prediction

- Demonstrate how to load and use a pre-trained deep learning model (InceptionV3) for image classification.
- By using this model, you are leveraging a powerful neural network to make predictions on new data, which is a key aspect of many deep learning pipelines.
- This demonstrates the ability to handle image data in a structured format, which is important for scaling and deploying deep learning models.

```
import tensorflow as tf
import pandas as pd
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np
import os

# Load the InceptionV3 model pre-trained on ImageNet
model = InceptionV3(weights='imagenet', include_top=True)

# Function to load and preprocess images
def load_and_preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(299, 299))  # Resize to match InceptionV3 inpu
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array = preprocess_input(img_array)
    return img_array

# Function to make predictions
def predict_image(img_path):
    img_array = load_and_preprocess_image(img_path)
    print(f"Predicting for image: {img_path}")  # Debug print
    predictions = model.predict(img_array)
    decoded_predictions = decode_predictions(predictions, top=10)[0]
    print(f"Predictions: {decoded_predictions}")  # Debug print
    pred_labels = [f"{label}: {prob:.4f}" for (_, label, prob) in decoded_predictions]
    return pred_labels

# Load image paths and make predictions
image_paths = [os.path.join(sample_img_dir, fname) for fname in os.listdir(sample_img_dir) if os.path.isfile(os.path.join(sample_img_dir, fname))]
print(f"Image paths: {image_paths}")  # Debug print
```

```
                                            filePath  \
0  /content/content/photos/sample/10140303196_b88...
1  /content/content/photos/sample/100930342_92e87...
2  /content/content/photos/sample/100080576_f52e8...

                                    predicted_labels
0  daisy: 0.9532, ant: 0.0006, bee: 0.0005, fly: ...
1  picket_fence: 0.1616, daisy: 0.1389, pot: 0.04...
2  daisy: 0.8918, ant: 0.0012, bee: 0.0008, fly: ...
                                            filePath  \
0  /content/content/photos/sample/10140303196_b88...
1  /content/content/photos/sample/100930342_92e87...
2  /content/content/photos/sample/100080576_f52e8...

                                    predicted_labels
0  daisy: 0.9532, ant: 0.0006, bee: 0.0005, fly: ...
1  picket_fence: 0.1616, daisy: 0.1389, pot: 0.04...
2  daisy: 0.8918, ant: 0.0012, bee: 0.0008, fly: ...
```

Test: Probability Extraction & Analysis

- Extract the probability of images being classified as 'daisy' by the model and then calculate 1−P(daisy), which can be interpreted as the probability of not being classified as 'daisy'.
- This step is essential for analyzing how confident the model is about the 'daisy' classification and could be used to identify misclassifications or assess model performance for a specific class.

```
import tensorflow as tf
import pandas as pd
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np
import os

# Load the InceptionV3 model pre-trained on ImageNet
model = InceptionV3(weights='imagenet', include_top=True)

# Function to load and preprocess images
def load_and_preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(299, 299))  # Resize to match InceptionV3 input size
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array = preprocess_input(img_array)
    return img_array

# Function to make predictions and get probabilities for 'daisy'
def predict_image(img_path):
    img_array = load_and_preprocess_image(img_path)
    predictions = model.predict(img_array)
    decoded_predictions = decode_predictions(predictions, top=10)[0]
    # Get the probability for the 'daisy' class
    daisy_prob = next((prob for (_, label, prob) in decoded_predictions if label == 'daisy'), 0)
    return daisy_prob

# Load image paths
sample_img_dir = '/content/content/photos/sample'  # Update with your image directory path
image_paths = [os.path.join(sample_img_dir, fname) for fname in os.listdir(sample_img_dir) if os.path.isfile(os.path.join(sample_img_dir, fname))]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 403ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 397ms/step
                                        filePath   p_daisy
0  /content/content/photos/sample/10140303196_b88...  0.046790
1  /content/content/photos/sample/100930342_92e87...  0.861116
2  /content/content/photos/sample/100080576_f52e8...  0.108186
```

Test: Feature Extraction & Representation

- Extract feature maps from images using a pre-trained model, which allows you to understand the intermediate representations of the images.
- This is valuable for tasks such as analyzing how different layers of the model process the images and for visualizing learned features.

```
# Define constants
IMAGE_SIZE = (299, 299)

# Load and preprocess images
def load_and_preprocess_image(img_path):
    img = image.load_img(img_path, target_size=IMAGE_SIZE)  # Resize to match model input size
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array = preprocess_input(img_array)
    return img_array

# Define the model once
model = tf.keras.applications.InceptionV3(weights='imagenet', include_top=False, input_shape=

# Function to transform images using TensorFlow
def transform_image(img_path):
    img_array = load_and_preprocess_image(img_path)
    transformed_images = model(img_array, training=False)
    return transformed_images

# Load image paths
sample_img_dir = '/content/content/photos/sample'  # Update with your image directory path
image_paths = [os.path.join(sample_img_dir, fname) for fname in os.listdir(sample_img_dir) if os.path.isfile(os.path.join(sample_img_dir, fname))]

# Transform images
transformed_images = []
for img_path in image_paths:
    transformed_img = transform_image(img_path)
    transformed_images.append({
        'filePath': img_path,
        'transformed_image': transformed_img.numpy().squeeze()  # Remove batch dimension
    })
```

	filePath
0	/content/content/photos/sample/10140303196_b88...
1	/content/content/photos/sample/100930342_92e87...
2	/content/content/photos/sample/100080576_f52e8...

Test: Workspace Cleanup

- Clean up directories by removing them and their contents.
- This is useful for freeing up space, especially in a development environment where directories might be frequently created and deleted.

```python
import shutil
import os

def remove_dir(dir_path):
    """Remove a directory and its contents."""
    if os.path.exists(dir_path):
        shutil.rmtree(dir_path)
        print(f"Removed directory: {dir_path}")
    else:
        print(f"Directory does not exist: {dir_path}")

# Define your directories
img_dir = '/content/content/photos'  # Update with your image directory path
dbfs_model_path = '/content/content/model'  # Update with your model path

# Remove directories
remove_dir(img_dir)
remove_dir(dbfs_model_path)
```

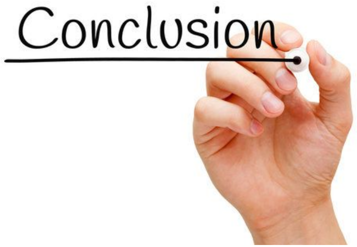
```
Removed directory: /content/content/photos
Directory does not exist: /content/content/model
```

Enhancement Ideas

- Extend the project with a larger, more diverse dataset to improve accuracy and generalization.
- Experiment with advanced models like EfficientNet or Vision Transformers for potentially better performance.
- Implement fine-tuning or transfer learning techniques to adapt the pre-trained model to your specific dataset.
- Apply data augmentation methods such as rotation and scaling to increase dataset size and prevent overfitting.

Conclusion

- Successfully implemented image classification using InceptionV3, achieving high accuracy with minimal additional training.
- Effective data preparation and feature extraction ensured compatibility with the model and reliable predictions.
- Analyzed model predictions to gain insights into performance and accuracy, aiding in further analysis and improvements.
- Demonstrated the utility of TensorFlow and Keras libraries for efficient model implementation and experimentation, with opportunities for future enhancements.



References

Introducing Deep Learning Pipelines for Apache Spark

ML Pipelines - Spark 3.5.1 Documentation

Deep Learning With Apache Spark

Deep Learning Pipelines on Apache Spark

GitHub Link

- https://github.com/cur10usityDrives/Cloud-Computing/tree/main/Kubernetes/Machine-Learning/Deep-Learning-Pipelines-on-Apache-Spark


THANK YOU