Question: Convert the batch image classification to a real-time execution model using Spark Streaming.

Submitted by Shramana Sinha, 23f1002703

1. Setup

1.1 DataProc Cluster configuration

A Google Cloud Dataproc cluster was created with the following configuration:

- **Master Node:** 1 (e2-standard-2)
- **Worker Nodes:** 2 (e2-standard-2)
- **Boot Disk Size:** 100GB per node
- **Operating System:** Ubuntu 22 with Dataproc image version 2.2
- **Networking:** Public IP address enabled
- **Installed Packages:** PyTorch 2.6.0, TorchVision 0.21.0, TensorFlow 2.19.0, Pillow 11.1.0

The cluster was provisioned using the following command in the google cloud shell:

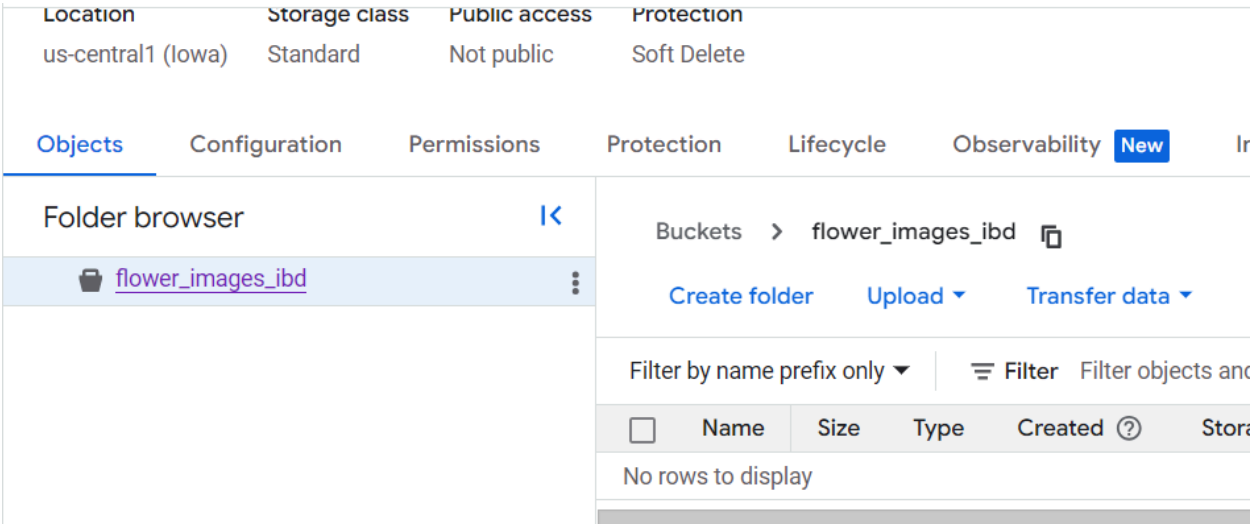
```
Unset
gcloud dataproc clusters create cluster-5a67 --enable-component-gateway
--region us-central1 --master-machine-type e2-standard-2
--master-boot-disk-size 100 --num-workers 2 --worker-machine-type e2-standard-2
--worker-boot-disk-size 100 --image-version 2.2-ubuntu22 --project
celtic-guru-448518-f8 --public-ip-address
--properties=^#^dataproc:pip.packages='torch==2.6.0,torchvision==0.21.0,tensorf
low==2.19.0,pillow==11.1.0'
```

1.2 GCS Setup

A GCS bucket named `flower_images_ibd` was created to store image data.

Region	us-central1
Zone	us-central1-c
Image version	2.2.50-ubuntu22
Autoscaling	Off
Performance Enhancements	
Advanced optimizations	Off
Advanced execution layer	Off
Google Cloud Storage caching	Off
Dataproc Metastore	None
Scheduled deletion	Off
Confidential computing enabled?	Disabled
Master node	Standard (1 master, N workers)
Machine type	e2-standard-2
Number of GPUs	0
Primary disk type	pd-standard
Primary disk size	100GB
Local SSDs	0
Worker nodes	2
Machine type	e2-standard-2
Number of GPUs	0
Primary disk type	pd-standard
Primary disk size	100GB

Screenshot: The dataproc cluster for the assignment



Screenshot: The gcs bucket for the assignment

2. Implementation

2.1 Image Streaming Component

2.1.1 Dataset Acquisition and Preparation

The implementation downloads the standard TensorFlow flower dataset, which consists of images categorized into multiple flower classes. The dataset is downloaded and extracted using:

```python
print(f"Downloading from: {download_url}")
urllib.request.urlretrieve(download_url, tgz_path)
print("Extracting dataset...")
with tarfile.open(tgz_path, "r:gz") as tar:
   tar.extractall()
os.remove(tgz_path)
```

The extracted dataset maintains its subdirectory structure, which is preserved during the streaming process:

```python
for category_path in pathlib.Path(self.data_dir).glob("*"):
    if category_path.is_dir():
        category = category_path.name
        jpg_paths = list(category_path.glob("*.jpg"))

        # Only add category if it has images
        if jpg_paths:
            image_paths[category] = jpg_paths
            print(f"Found {len(jpg_paths)} images in {category} category")
```

2.1.2 Streaming Implementation

The streaming process uploads images at regular intervals (5 seconds) to simulate a continuous data source. The streaming algorithm:

1. Flattens all image paths into a single list

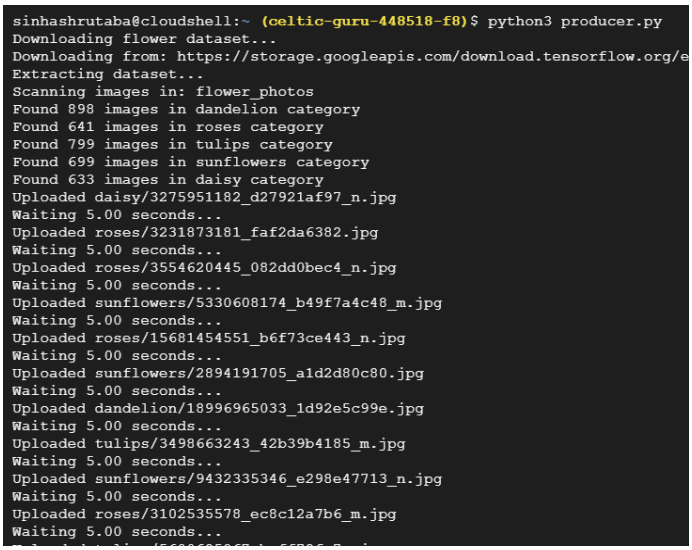
```python
all_images = []
for category, paths in self.image_paths.items():
    for path in paths:
        all_images.append((category, path))
```

2. Shuffles the images for randomized order

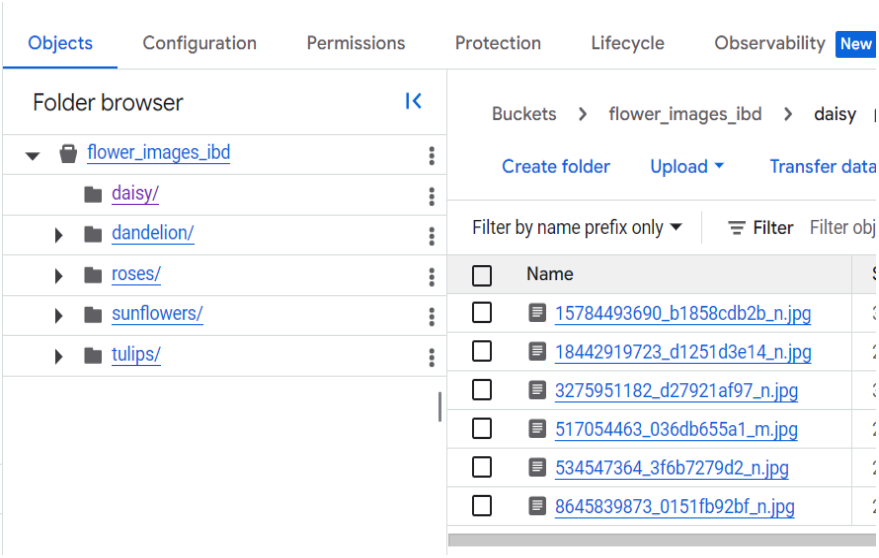
```python
random.shuffle(all_images)
```

3. Uploads each image to the GCS bucket preserving the original category structure

```python
blob_name = f"{category}/{image_path.name}"
blob = self.bucket.blob(blob_name)
blob.upload_from_filename(str(image_path))
```



Screenshot: The producer, running on google cloud shell, uploading images to the gcs bucket



Screenshot: The images getting uploaded in the gcs bucket

2.2 Real-Time Classification Component

2.2.1 Spark Session Configuration

To avoid permission issues while model downloading, the Spark session is configured to set the `TORCH_HOME` directory to `/tmp/torch_cache`.

```python
def create_spark_session():
    return (
        SparkSession.builder.appName("RealTimeImageClassification")
        .config("spark.executorEnv.TORCH_HOME", "/tmp/torch_cache")
        .config("spark.streaming.stopGracefullyOnShutdown", "true")
        .getOrCreate()
    )
```

2.2.2 Image Processing Pipeline

The implementation creates a custom PyTorch dataset class that handles the transformation of binary image content into tensor form suitable for the MobileNetV2 model:

```python
def _preprocess(self, content):
    image = Image.open(io.BytesIO(content))
    transform = transforms.Compose(
        [
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(
                mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
            ),
        ]
    )
    return transform(image)
```

The preprocessing steps are:

1. Resize images to 256px
2. Center crop to 224px
3. Convert to tensor format
4. Apply channel-wise normalization with means and standard deviations

2.2.3 Model Integration with Pandas UDF

To integrate the PyTorch model with Spark Streaming, a Pandas UDF is implemented:

```python
def imagenet_model_udf(model_fn):
   def predict(content_series_iter):
        model = model_fn()
        model.eval()
        for content_series in content_series_iter:
            dataset = ImageNetDataset(list(content_series))
            loader = DataLoader(dataset)
            with torch.no_grad():
                for image_batch in loader:
                    predictions = model(image_batch).numpy()
                    predicted_labels = [
                        x[0] for x in decode_predictions(predictions, top=1)
                    ]
                    yield pd.DataFrame(predicted_labels)

    return_type = "class: string, desc: string, score:float"
    return pandas_udf(return_type, PandasUDFType.SCALAR_ITER)(predict))
```

This UDF:

1. Loads the MobileNetV2 model only once per executor
2. Uses TensorFlow's `decode_predictions` function to convert raw outputs to human-readable labels
3. Returns a structured result with class name, class description and score.

2.2.4 Streaming Query Configuration

The Spark Structured Streaming query is configured to:

1. Read binary files from the GCS bucket and process one file per micro-batch:

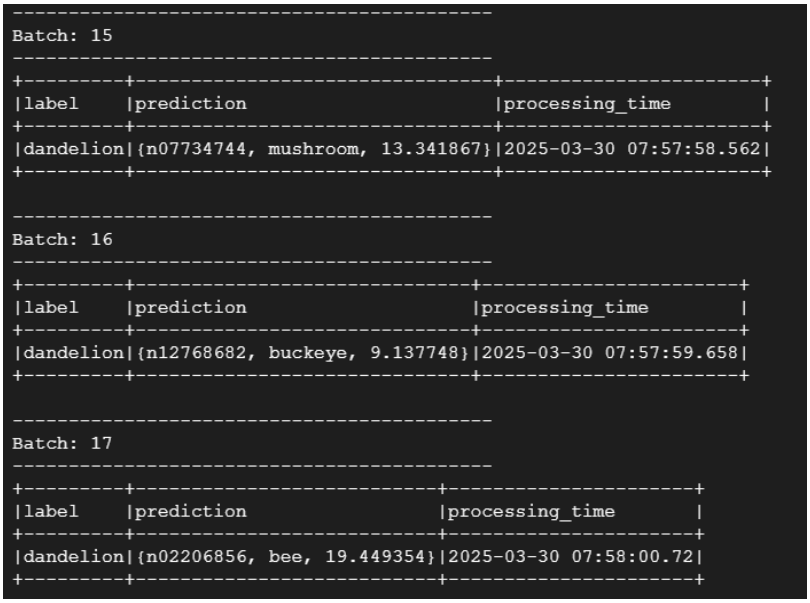
```python
stream_df = (
        spark.readStream.format("binaryFile")
        .schema(input_schema)
        .option("maxFilesPerTrigger", 1)
        .load(input_path)
    )
```

2. Extract the label from the file path and apply the model for predictions

```python
classified_stream = (
        stream_df.withColumn("prediction", mobilenet_v2_udf(col("content")))
        .withColumn("label", extract_label(col("path")))
        .withColumn("processing_time", current_timestamp())
        .select("label","prediction", "processing_time",)
    )
```

3. Output results to the console in append mode

```python
query = (
        classified_stream.writeStream.outputMode("append")
        .format("console")
        .option("truncate", "false")
        .start()
    )
```



Screenshot: The streaming output of the model prediction