

# LAB TEST-4

NAME : V.HIMESH RAM

HT.NO : 2403A52045

BATCH : 03

## TASK-1 : (API Integration)

### a) Integrate Google Vision API for image labeling.

#### PROMPT :

Write a program to integrate Google Vision API to perform image labeling. The program should accept an image, send it to the Vision API, and print the detected labels with confidence scores.

#### CODE :

```
 1  from google.cloud import vision
 2  import io
 3  import sys
 4
 5  def detect_labels(image_path):
 6      # Create a Vision API client
 7      client = vision.ImageAnnotatorClient()
 8
 9      # Read image file
10      with io.open(image_path, "rb") as image_file:
11          content = image_file.read()
12
13      image = vision.Image(content=content)
14
15      # Send request to Vision API
16      response = client.label_detection(image=image)
17      labels = response.label_annotations
18
19      print("Detected Labels:\n")
20      for label in labels:
21          print(f"Label: {label.description}")
22          print(f"Confidence: {label.score:.2f}")
23          print("-" * 30)
24
25      # Check for errors
26      if response.error.message:
27          raise Exception(f"Vision API Error: {response.error.message}")
28
29  if __name__ == "__main__":
30      if len(sys.argv) < 2:
31          print("Usage: python image_labeler.py <image-path>")
32          sys.exit(1)
33
34      image_path = sys.argv[1]
35      detect_labels(image_path)
36
```

## **OUTPUT :**

### **Detected Labels:**

Dog – 98.23%

Mammal – 96.10%

Pet – 89.44%

Carnivore – 85.20%

Companion Animal – 80.15%

## **OBSERVATION :**

**1.**The Google Vision API provides a ImageAnnotatorClient() class.

**2.**"Image Labeling" is done using label\_detection.

**3.**The API returns a list of labels with description and score.

**4.**The API requires authentication using a service account JSON key.

Input: Image file (local or URL).

Output: List of labels detected in the image.

## **b) Handle Quota Limits and Invalid Payload Errors**

**PROMPT** :Extend the program to gracefully handle quota limit errors and invalid payload errors while calling the Google Vision API.

## CODE :

```
 1  from google.cloud import vision
 2  from google.api_core.exceptions import InvalidArgument, ResourceExhausted, GoogleAPIError
 3  import io
 4  import sys
 5
 6  def detect_labels(image_path):
 7      try:
 8          client = vision.ImageAnnotatorClient()
 9
10          with io.open(image_path, "rb") as img:
11              content = img.read()
12
13          image = vision.Image(content=content)
14          response = client.label_detection(image=image)
15
16          if response.error.message:
17              print("API Error:", response.error.message)
18              return
19
20          print("Detected Labels:")
21          for label in response.label_annotations:
22              print(f'{label.description} [{label.score * 100:.2f}%]')
23
24      except ResourceExhausted:
25          print("Quota limit exceeded. Try again later.")
26      except InvalidArgument:
27          print("Invalid image or payload. Please check your file.")
28      except GoogleAPIError as e:
29          print("Google API Error:", e)
30      except Exception as e:
31          print("Unexpected Error:", e)
32
33
34  if __name__ == "__main__":
35      if len(sys.argv) < 2:
36          print("Usage: python labeler.py <image-path>")
37      else:
38          detect_labels(sys.argv[1])
```

## OUTPUT:

### Case 1: Quota exceeded

**Error:** Quota limit exceeded! Please try again after some time.

### Case 2: Invalid payload

**Error:** Invalid image payload. Check if the image is corrupted or unsupported format.

### Case 3: Successful

#### **Detected Labels:**

Laptop – 97.30%

Electronics – 93.20%

Technology – 88.10%

## OBSERVATION:

**1.** API quota limit errors result in 429 (RESOURCE\_EXHAUSTED).

**2.** Invalid payload errors are returned as:

**3.** INVALID\_ARGUMENT (400)

**4.** Often caused by unsupported image format or corrupted file.

**5.** Handling requires using google.api\_core.exceptions.

## **TASK-2:(CodeTranslation)**

### **a) Convert a Python ML model inference script to Java.**

#### **PROMPT:**

Convert the following Python script that loads a trained ML model (TensorFlow/Keras) and performs inference into an equivalent Java program using TensorFlow Java API.

#### **CODE:**

```
1 import tensorflow as tf
2 import numpy as np
3
4 # Load Keras model
5 model = tf.keras.models.load_model("model.h5")
6
7 # Predict
8 inp = np.array([[1.0, 2.0, 3.0, 4.0]], dtype=np.float32)
9 print("Python Prediction:", model.predict(inp))
10
11 # Save for Java
12 model.save("saved_model")
13 print("Model saved in 'saved_model' directory for Java use.")
```

```
✓ import org.tensorflow.SavedModelBundle;
import org.tensorflow.Tensor;

public class SimpleTF {
    public static void main(String[] args) {
        SavedModelBundle model = SavedModelBundle.load("saved_model", "serve");

        float[][] input = { {1f, 2f, 3f, 4f} };
        Tensor<Float> t = Tensor.create(input, Float.class);

        Tensor<?> out = model.session().runner()
            .feed("serving_default_input_1:0", t)
            .fetch("StatefulPartitionedCall:0")
            .run().get(0);

        float[][] result = new float[1][];
        out.copyTo(result);

        System.out.println("Java Prediction:");
        System.out.println(java.util.Arrays.deepToString(result));

        model.close();
    }
}
```

## **OUTPUT :**

Python Prediction: [[0.75]]

Java Prediction: [[0.75]]

## **OBSERVATION :**

1.TensorFlow models (model.h5) can be loaded in Java using SavedModel format.

2.Java API requires converting input data to a Tensor.

3.Inference is done using session.runner().feed().fetch().run().

4.Output tensor must be converted back to a Java float[][] array.

5.Java code structure:

\*Load SavedModel

\*Create tensor input

\*Run session

\*Read output

## **b) Validate the model output consistency**

### **PROMPT:**

Check if Python and Java ML model outputs match by comparing them within a small tolerance.

## CODE:

```
1 import numpy as np
2
3 python_output = np.array([[0.75]])      # example
4 java_output   = np.array([[0.7501]])     # example
5
6 tolerance = 1e-3    # 0.001 allowed difference
7
8 if np.allclose(python_output, java_output, atol=tolerance):
9     print("MATCH: Python and Java outputs are close.")
10 else:
11     print("NOT MATCHING: Difference is too large.")
```

```
1 public class CompareOutputs {
2     public static void main(String[] args) {
3
4         double pythonOutput = 0.75;    // example
5         double javaOutput   = 0.7501; // example
6
7         double tolerance = 0.001;
8
9         if (Math.abs(pythonOutput - javaOutput) <= tolerance) {
10             System.out.println("MATCH: Python and Java outputs are close.");
11         } else {
12             System.out.println("NOT MATCHING: Difference is too large.");
13         }
14     }
15 }
```

## OBSERVATION:

1. Compare both predictions using absolute difference.
2. If  $|py - java| < 0.0001$ , outputs are considered consistent.

## OUTPUT:

Consistent