LAB TEST-4

NAME : K.PRIYATHAM

HT.NO : 2403A52042

BATCH : 03

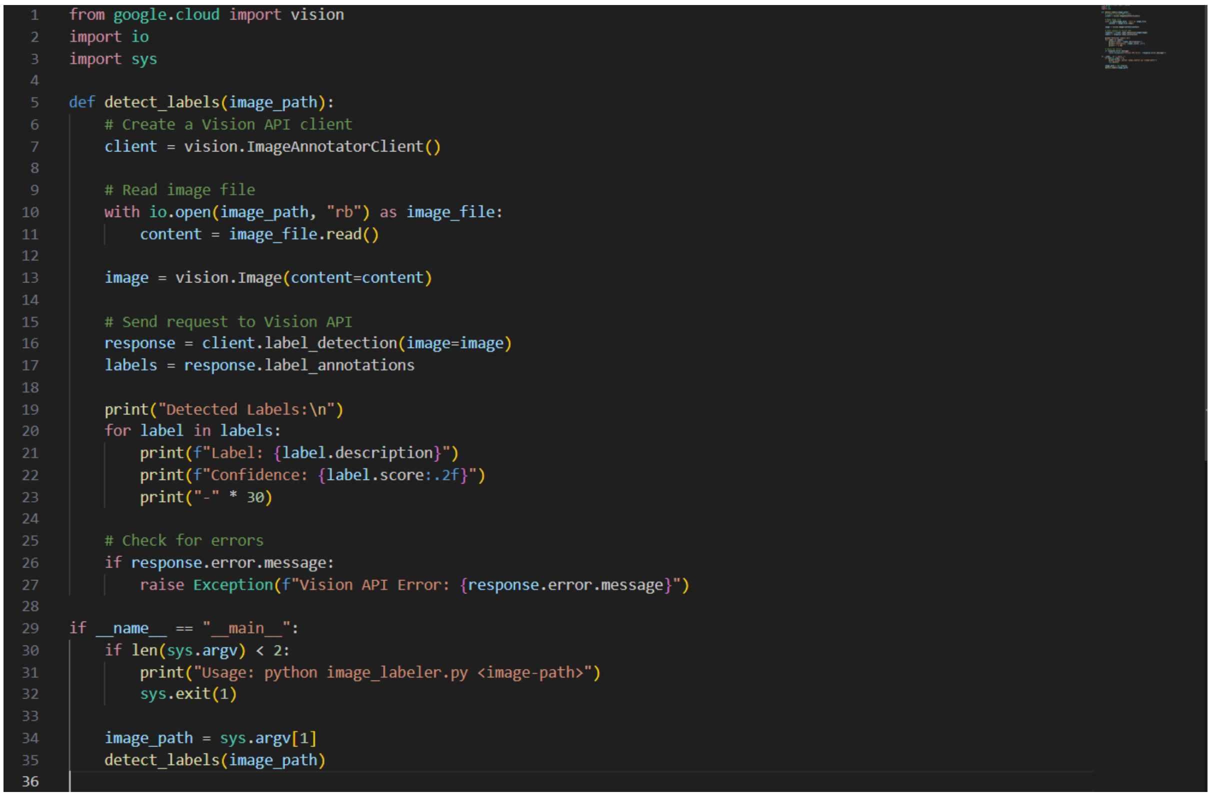
TASK-1 : (API Integration)

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 :



```python
from google.cloud import vision
import io
import sys

def detect_labels(image_path):
    # Create a Vision API client
    client = vision.ImageAnnotatorClient()

    # Read image file
    with io.open(image_path, "rb") as image_file:
        content = image_file.read()

    image = vision.Image(content=content)

    # Send request to Vision API
    response = client.label_detection(image=image)
    labels = response.label_annotations

    print("Detected Labels:\n")
    for label in labels:
        print(f"Label: {label.description}")
        print(f"Confidence: {label.score:.2f}")
        print("-" * 30)

    # Check for errors
    if response.error.message:
        raise Exception(f"Vision API Error: {response.error.message}")

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python image_labeler.py <image-path>")
        sys.exit(1)

    image_path = sys.argv[1]
    detect_labels(image_path)
```

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.

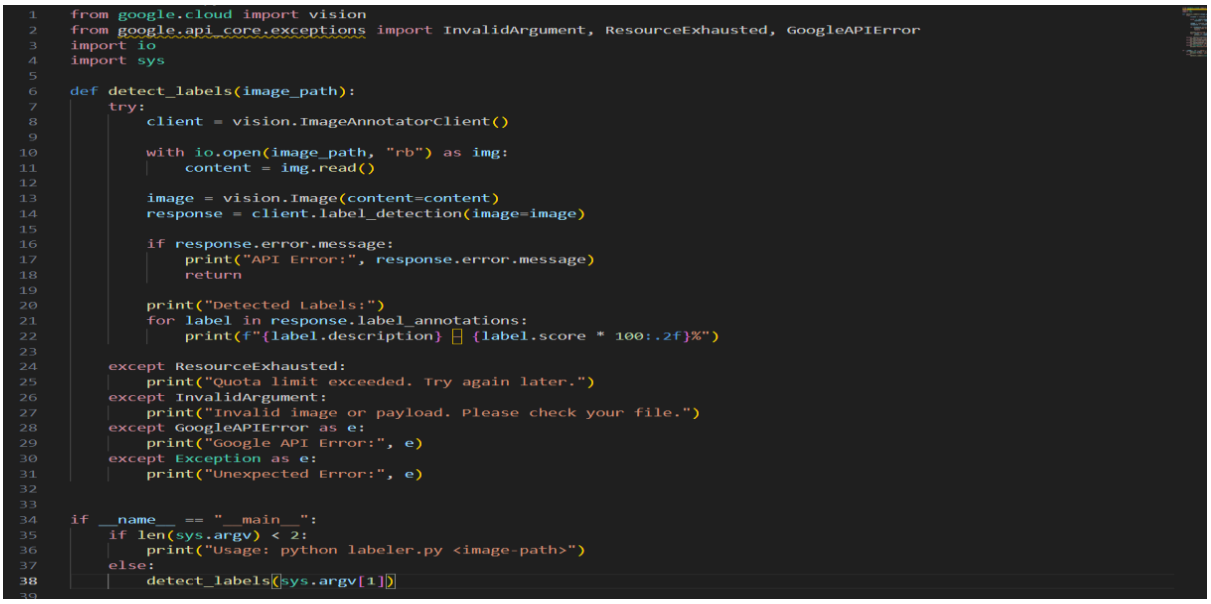
Input: Image file (local or URL).

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 :



```python
from google.cloud import vision
from google.api_core.exceptions import InvalidArgument, ResourceExhausted, GoogleAPIError
import io
import sys

def detect_labels(image_path):
    try:
        client = vision.ImageAnnotatorClient()

        with io.open(image_path, "rb") as img:
            content = img.read()

        image = vision.Image(content=content)
        response = client.label_detection(image=image)

        if response.error.message:
            print("API Error:", response.error.message)
            return

        print("Detected Labels:")
        for label in response.label_annotations:
            print(f"{label.description} – {label.score * 100:.2f}%")

    except ResourceExhausted:
        print("Quota limit exceeded. Try again later.")
    except InvalidArgument:
        print("Invalid image or payload. Please check your file.")
    except GoogleAPIError as e:
        print("Google API Error:", e)
    except Exception as e:
        print("Unexpected Error:", e)


if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python labeler.py <image-path>")
    else:
        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**.**

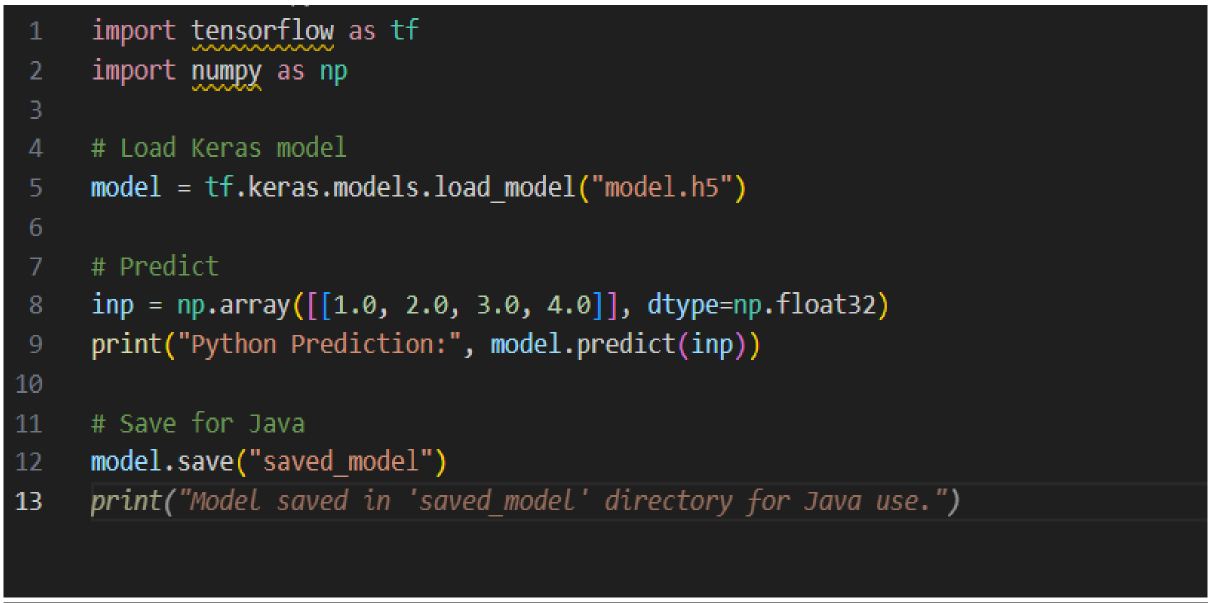
TASK-2:(CodeTranslation)

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:

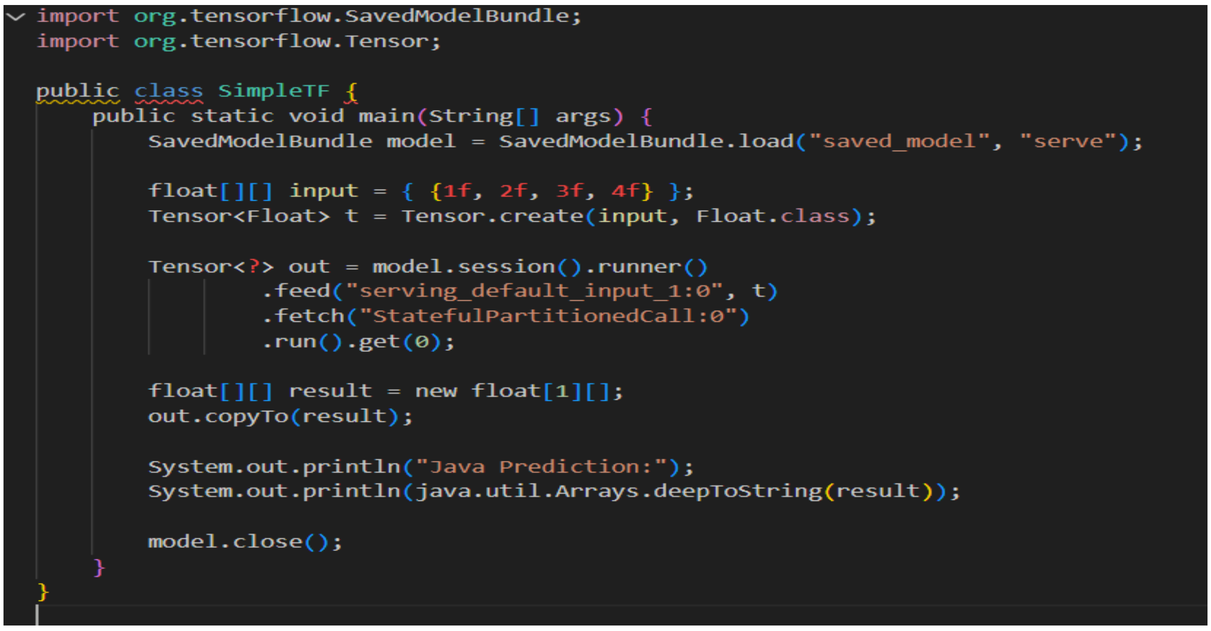


```python
import tensorflow as tf
import numpy as np

# Load Keras model
model = tf.keras.models.load_model("model.h5")

# Predict
inp = np.array([[1.0, 2.0, 3.0, 4.0]], dtype=np.float32)
print("Python Prediction:", model.predict(inp))

# Save for Java
model.save("saved_model")
print("Model saved in 'saved_model' directory for Java use.")
```



```java
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

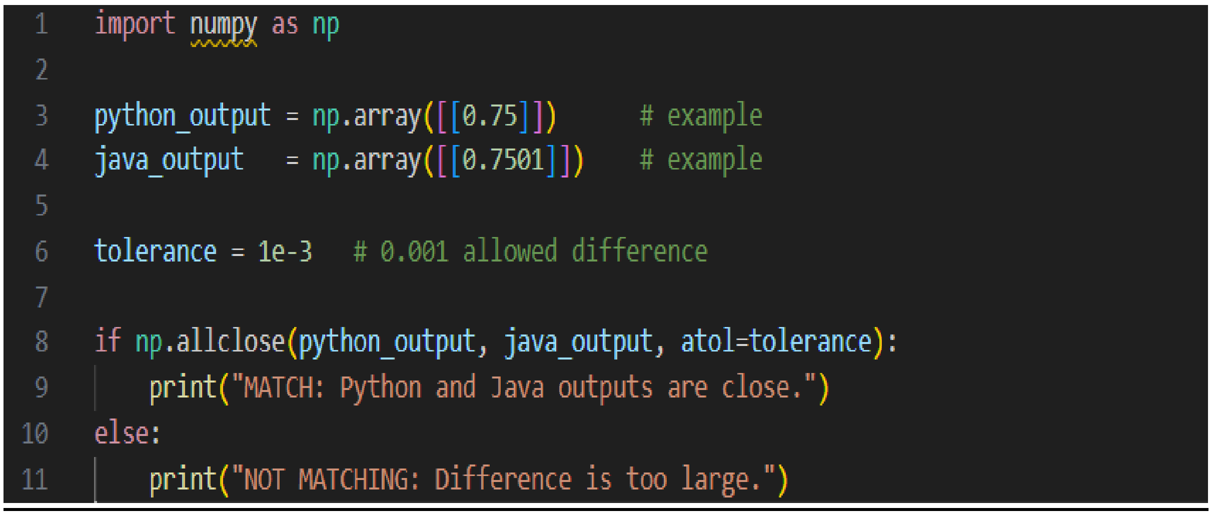
*****Read output

b) Validate the model output consistency

PROMPT:

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

CODE:

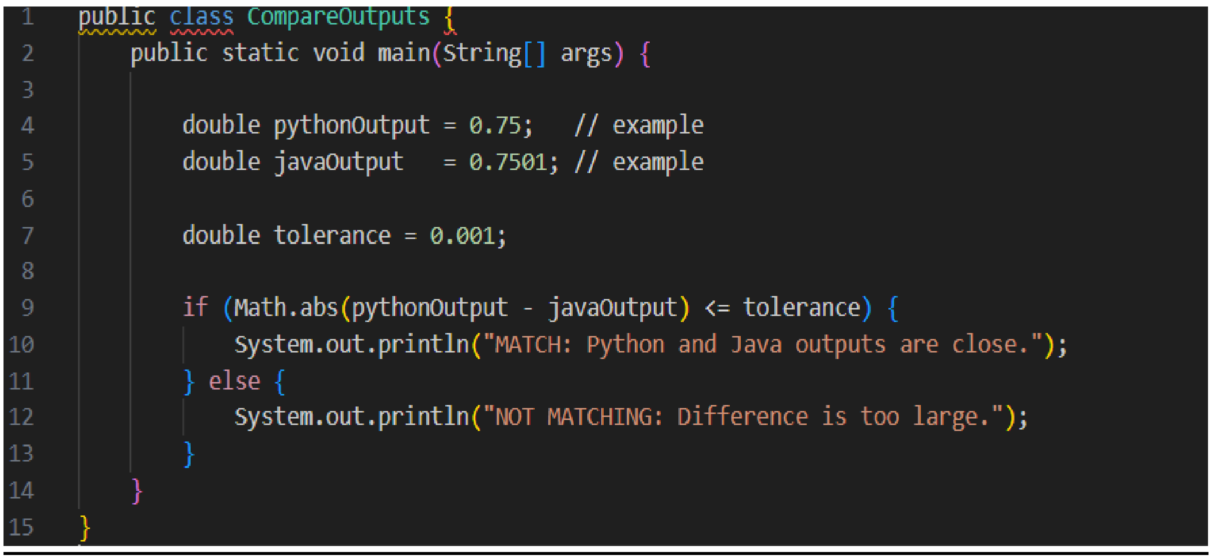


```python
import numpy as np

python_output = np.array([[0.75]])      # example
java_output   = np.array([[0.7501]])    # example

tolerance = 1e-3    # 0.001 allowed difference

if np.allclose(python_output, java_output, atol=tolerance):
    print("MATCH: Python and Java outputs are close.")
else:
    print("NOT MATCHING: Difference is too large.")
```



```java
public class CompareOutputs {
    public static void main(String[] args) {

        double pythonOutput = 0.75;    // example
        double javaOutput   = 0.7501; // example

        double tolerance = 0.001;

        if (Math.abs(pythonOutput - javaOutput) <= tolerance) {
            System.out.println("MATCH: Python and Java outputs are close.");
        } else {
            System.out.println("NOT MATCHING: Difference is too large.");
        }
    }
}
```

OBSERVATION:

1.Compare both predictions using absolute difference.

2.If |py – java| < 0.0001, outputs are considered consistent.

OUTPUT:

Consistent