# DEPARTMENT OF INFORMATION TECHNOLOGY

|  |  |
| --- | --- |
| **COURSE CODE: DJ19ITL504** | **DATE:** |
| **COURSE NAME: Artificial Intelligence Laboratory** | **CLASS: TY-IT** |
| **Name: Atharva Narvekar** | **Rollno:I024** |

**EXPERIMENT NO.10**

**CO/LO:** Apply various AI approaches to knowledge intensive problem solving, reasoning, planning and uncertainty.

**AIM / OBJECTIVE:** Implement Explainable AI for image and text

# Code:

# Step 1: Import necessary libraries from transformers import pipeline import numpy as np import matplotlib.pyplot as plt from lime.lime\_text import LimeTextExplainer

# Step 2: Load a pre-trained text classification model using BERT classifier = pipeline('text-classification')

# Step 3: Read a text file or use a string input for classification # Replace 'input.txt' with your own text file

name file\_path = 'input.txt' with open(file\_path, 'r') as f:

input\_text = f.read()

# Step 4: Display the uploaded text print("Uploaded Text:\n", input\_text)

# Step 5: Classify the text using the pre-trained BERT model predictions = classifier(input\_text) print("\nPredictions:", predictions)

# Step 6: Display the top classification label top\_prediction = predictions[0]

print("\nTop Prediction: {} with confidence {:.2f}".format(top\_prediction['label'], top\_prediction['score']))

# Step 7: LIME (Local Interpretable Model-agnostic Explanations) for Text Explainability # Define class names according to your classification labels

explainer = LimeTextExplainer(class\_names=['LABEL\_0', 'LABEL\_1']) # Modify based on your classes

# Define a prediction function for LIME to work with BERT def predict\_proba(texts):

results = classifier(texts)

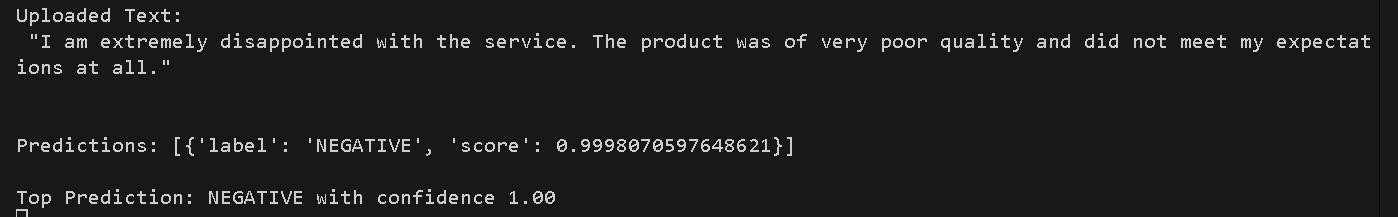
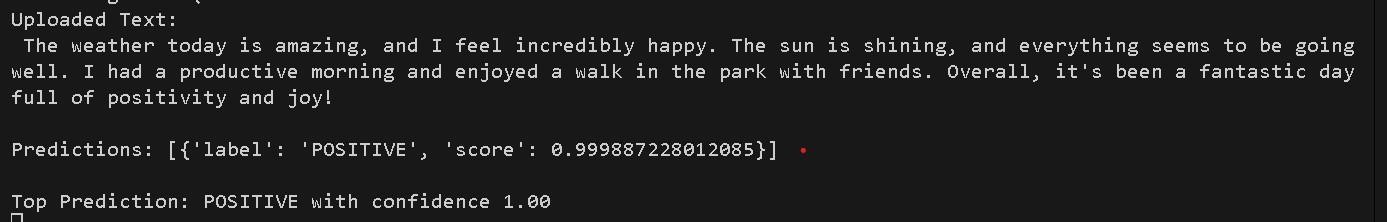
# Convert the classifier output to a probability-like format (required for LIME) probabilities = [] for result in results:

probabilities.append([result['score'], 1 - result['score']]) # Adjust for binary classification return np.array(probabilities)

# Step 8: Explain the prediction using LIME

explanation = explainer.explain\_instance(input\_text, predict\_proba, num\_features=10)

# Step 9: Visualize the explanation (important words/phrases for classification) explanation.show\_in\_notebook(text=True) # Will open the explanation in a Jupyter Notebook

Output:

# Code:

import os import keras

from keras.applications import inception\_v3 as inc\_net from keras.preprocessing import image

from keras.applications.imagenet\_utils import decode\_predictions from skimage.io import imread import matplotlib.pyplot as plt import numpy as np import lime

from lime import lime\_image

from skimage.segmentation import mark\_boundaries print('Notebook run using keras:', keras. version )

# Load the InceptionV3 model pre-trained on ImageNet inet\_model

= inc\_net.InceptionV3()

def transform\_img\_fn(path\_list): out = [] for img\_path

in path\_list:

img = image.load\_img(img\_path, target\_size=(299, 299)) x = image.img\_to\_array(img) x = np.expand\_dims(x,

axis=0) x = inc\_net.preprocess\_input(x) out.append(x)

return np.vstack(out)

# Replace this path with your actual image path image\_path = r'C:\Users\dhruv\OneDrive\Desktop\dwn\image.png' images = transform\_img\_fn([image\_path])

# Visualize the image (undoing the preprocessing normalization) plt.imshow(images[0] / 2 + 0.5) plt.show()

# Predict with the model preds = inet\_model.predict(images) for x in decode\_predictions(preds, top=5)[0]:

print(x)

# Initialize the Lime Image Explainer explainer

= lime\_image.LimeImageExplainer()

# Get the explanation explanation = explainer.explain\_instance( images[0].astype('double'), inet\_model.predict,

top\_labels=5, hide\_color=0, num\_samples=1000

)

# Get the image with mask and display it temp, mask = explanation.get\_image\_and\_mask( explanation.top\_labels[0], positive\_only=True, num\_features=5, hide\_rest=True

)

# Correct the visualization scaling for plotting plt.imshow(mark\_boundaries(temp

/ 2 + 0.5, mask)) plt.show()

temp, mask = explanation.get\_image\_and\_mask(explanation.top\_labels[0], positive\_only=False, num\_features=10, hide\_rest=False) plt.imshow(mark\_boundaries(temp / 2 + 0.5, mask)) plt.show() temp, mask = explanation.get\_image\_and\_mask(explanation.top\_labels[0], positive\_only=False, num\_features=1000, hide\_rest=False, min\_weight=0.05) plt.imshow(mark\_boundaries(temp / 2 + 0.5, mask)) plt.show()

#Select the same class explained on the figures above. ind

= explanation.top\_labels[0]

#Map each explanation weight to the corresponding superpixel dict\_heatmap

= dict(explanation.local\_exp[ind])

heatmap = np.vectorize(dict\_heatmap.get)(explanation.segments)

#Plot. The visualization makes more sense if a symmetrical colorbar is used. plt.imshow(heatmap, cmap = 'RdBu', vmin = -heatmap.max(), vmax = heatmap.max()) plt.colorbar()

temp, mask = explanation.get\_image\_and\_mask(explanation.top\_labels[1], positive\_only=True, num\_features=6, hide\_rest=True) plt.imshow(mark\_boundaries(temp / 2 + 0.5, mask)) plt.show()

temp, mask = explanation.get\_image\_and\_mask(explanation.top\_labels[1], positive\_only=False, num\_features=5, hide\_rest=False) plt.imshow(mark\_boundaries(temp / 2 + 0.5, mask)) plt.show()

import json

from tensorflow.keras.applications.resnet50 import ResNet50, preprocess\_input import shap

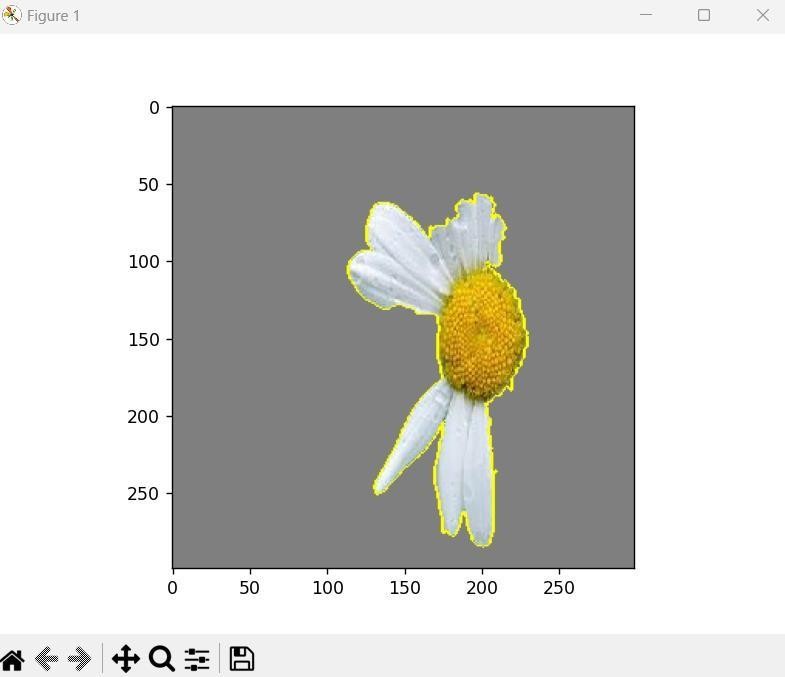
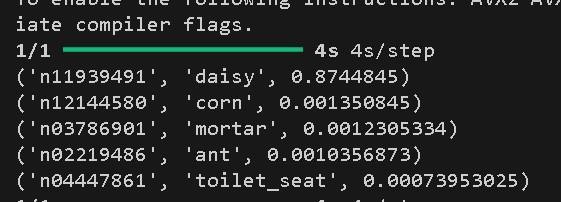
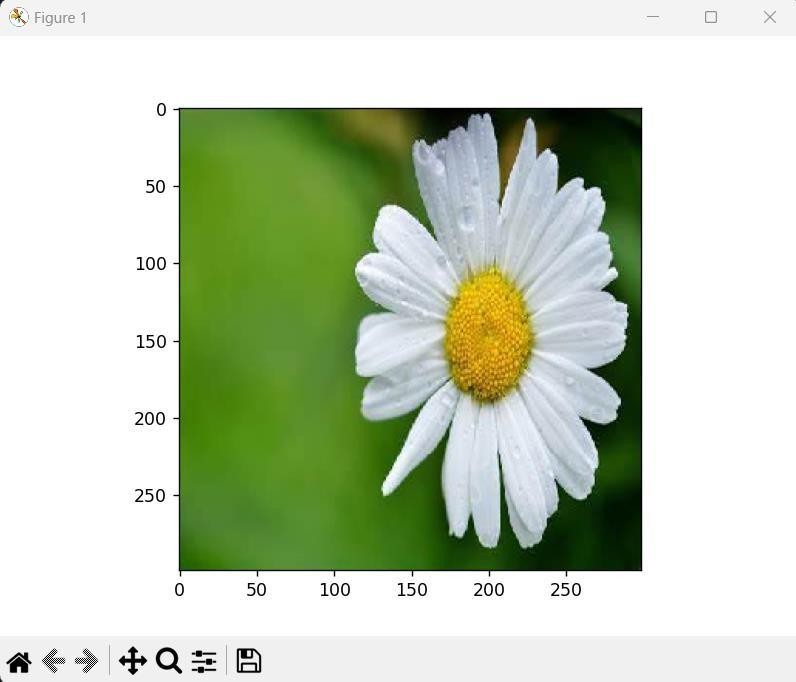
# load pre-trained model and data model

= ResNet50(weights="imagenet") X, y

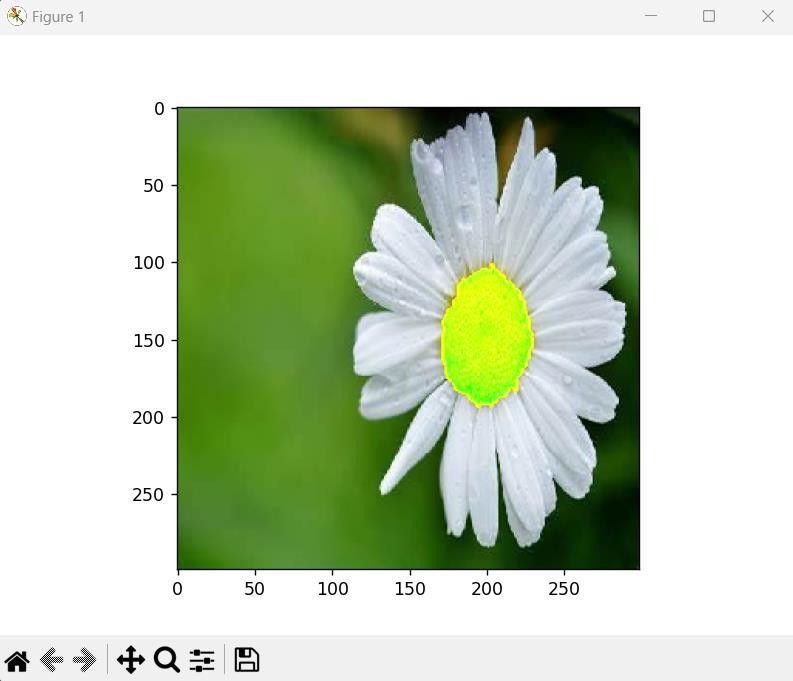
= shap.datasets.imagenet50() print(y) plt.imshow(X[20]) plt.show()

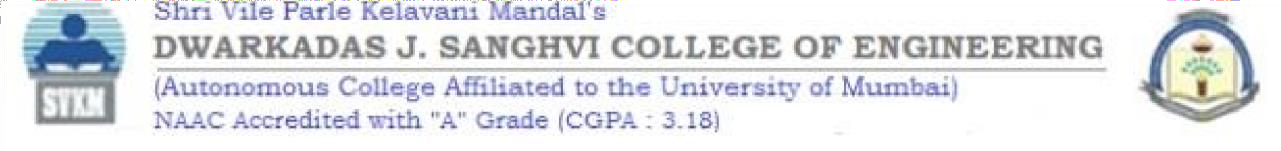
X = np.clip(X, 0, 255).astype(np.uint8) plt.imshow(X[4]) plt.show()

# Output:

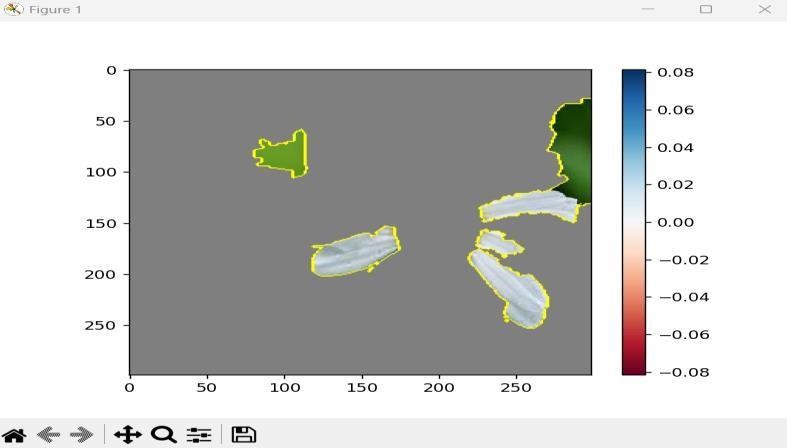


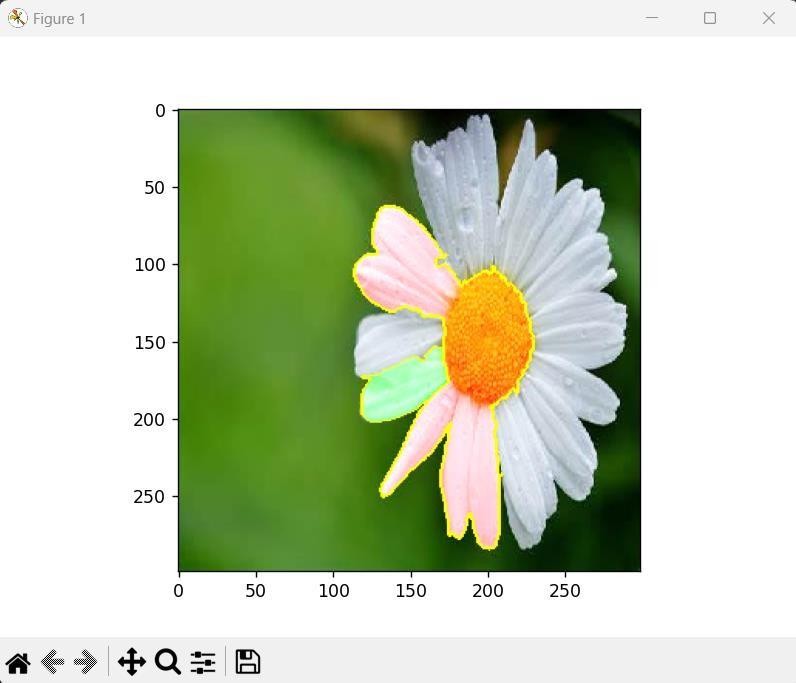






Academic Year 2023-24 SAP ID:





**CONCLUSION: We implemented the explainable AI. REFERENCES:**

[1] Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, 2nd Edition, Pearson Education, 2010