

Internship Report

MKU-RUSA

Particulars	Report Details
Name	PRIYADHARSHANA M
Department	Computer Science
College/University	Government Arts college,Melur
Course	M.Sc(computer science)-1st year
Mobile Number	6369598603
Email ID	Priyadharshanam12@gmail.com
Joining Date	23.05.2024
Supervising Faculty & Department at MKU	Dr.R.Rathinasabapathy(department of computer Application)
List of Objectives	Basic image processing tools,CNN using image classification, Thresholding functions tools,extract character from an image using threshold, Contour detection,water _shed segmentation,region_base segmentation, Canny_edge detection,use sift and histogram to find key point matches between images,object detection using already existing model And use linear regression to predict mobile price.
List the Other Learnings during this internship period	Numpy,pandas,matplotlib,tkinter,torchvision transformation, Torchvision model,seaborn,sklearn,tensorflow,keras and Basic image processing, segmentation techniques,cnn,linear regression, Object detection and sift-histogram.
How this experience would be useful for your career development	This will be help my final year projectAnd support my career in future.This is the first time I learn image processing and machine learning concept through this internship.Thank you for gave me this opportunity.

WEEK1

S.NO	TASKS
1	Learn and create basic image processing tools
2	Learn CNN and implement image classification
3	Learn threshold and create tools to apply function for various threshold functions for various images

WEEK2

S.NO	TASKS
4	Use threshold function to extract character from an image
5	Learn and implement contour segmentation technique
6	Learn and implement water_shed segmentation technique

WEEK3

S.NO	TASKS
7	Learn and implement region_base segmentation technique
8	Learn and implement canny_edge _detection segmentation technique

WEEK 4

S.NO	TASKS
9	Learn and implement shift feature and histogram to find keypoint matches between images
10	Object detection using already existing model like alexnet
11	Learn linear regression and predict mobile price

ABSTRACT

During my internship, I acquired foundational skills in image processing, segmentation techniques, Convolutional Neural Networks (CNNs), linear regression, and object detection using pre-trained models. My journey began with mastering basic image processing methods, essential for enhancing image quality and preparing data for subsequent analysis. I then explored basic segmentation techniques to divide digital images into meaningful segments, aiding in focused analysis of specific image regions.

In the realm of machine learning, I studied CNNs, which are pivotal for image recognition and classification tasks. Through practical projects, I built and trained CNN models, learning about their architecture and the significance of convolutional layers for feature extraction. I also delved into linear regression, a fundamental technique for predictive analysis, and its applications within machine learning.

This comprehensive learning experience provided me with a robust foundation in image processing, segmentation, and machine learning, equipping me with the skills necessary for advanced studies and future projects in these fields.

INDEX

S.NO	TASKS
TASK1	Basic image processing tool
TASK2	Use CNN for image classification
TASK3	Threshold functions tool
TASK4	Use threshold to extract character from an image
TASK5	Contour detection
TASK6	Water_shed segmentation technique
TASK7	Region_base segmentation technique
TASK8	Canny_edge detection
TASK9	Using sift to find keypoint matches between images
TASK10	Object detection using already existing model
TASK11	Learn linear regression and predict mobile price

TASK-1

AIM : LEARN AND CAREATE A BASIC IMAGE PROCESSING TOOL

OBJECTIVE

The objective of the above program is to create an interactive image processing tool with a graphical user interface (GUI) using Python. The tool allows users to:

1. **Upload an Image:** Select an image file from their local system.
2. **Perform Basic Image Operations:**
 - **Resize:** Change the dimensions of the image.
 - **Rotate:** Rotate the image by a specified angle.
 - **Translate:** Move the image by specified x and y offsets.
 - **Normalize:** Adjust the intensity values of the image.
 - **Edge Detection:** Detect edges in the image using the Canny edge detector.
 - **Blur:** Apply Gaussian blur to the image.
 - **Morphological Operations:** Perform operations like erosion, dilation, opening, and closing on the image.
3. **Display Processed Images:** Show the results of these operations in the GUI.
4. **Handle User Input Validation:** Validate user inputs (e.g., numeric values for dimensions, angles, kernel sizes) and display error messages if inputs are invalid.
5. **Enhance User Experience:** Provide a user-friendly interface with buttons, entry fields, and image display areas using Tkinter, facilitating intuitive interaction with image processing functionalities.

Overall, the objective is to create a versatile tool that enables users to experiment with different image processing techniques interactively, all within a single application window.

DESCRIPTION

Imports

```
import cv2
```

```
import numpy as np
```

```
from tkinter import filedialog, Tk, Button, Label, Entry, StringVar, OptionMenu, Canvas,
messagebox
```

```
from PIL import Image, ImageTk
```

- **cv2:** OpenCV library for image processing.
- **numpy:** Library for numerical operations in Python.
- **tkinter:** Python's standard GUI (Graphical User Interface) toolkit.
- **PIL:** Python Imaging Library, used here for handling images and image formats.

Global Variable

`uploaded_image = None`

- **uploaded_image**: Stores the image uploaded by the user using `cv2.imread`.

Function Definitions

`def upload_image():`

`global uploaded_image`

`Tk().withdraw()`

`filename = filedialog.askopenfilename()`

`if filename:`

`uploaded_image = cv2.imread(filename)`

`display_image(uploaded_image, "Uploaded Image")`

- **Tk().withdraw()**: Creates a hidden root window to facilitate file dialog.
- **filedialog.askopenfilename()**: Opens a file dialog for selecting an image file.
- **cv2.imread(filename)**: Reads the selected image file using OpenCV.
- **display_image**: Displays the uploaded image in the GUI.

Resize Image Function

`def resize_image():`

`if uploaded_image is None:`

`messagebox.showerror("Error", "Please upload an image first.")`

`return`

`try:`

`width = int(width_entry.get())`

`height = int(height_entry.get())`

`result = cv2.resize(uploaded_image, (width, height))`

```
display_image(result, "Resized Image")
```

```
except ValueError:
```

```
    messagebox.showerror("Error", "Please enter valid width and height.")
```

- **cv2.resize**: Resizes the uploaded image to the specified width and height.
- **width_entry** and **height_entry**: Tkinter Entry widgets to input width and height.
- **messagebox.showerror**: Shows an error message box if input values are invalid.

Rotate Image Function

```
def rotate_image():
```

```
    if uploaded_image is None:
```

```
        messagebox.showerror("Error", "Please upload an image first.")
```

```
        return
```

```
    try:
```

```
        angle = float(angle_entry.get())
```

```
        (h, w) = uploaded_image.shape[:2]
```

```
        center = (w // 2, h // 2)
```

```
        M = cv2.getRotationMatrix2D(center, angle, 1.0)
```

```
        result = cv2.warpAffine(uploaded_image, M, (w, h))
```

```
        display_image(result, "Rotated Image")
```

```
    except ValueError:
```

```
        messagebox.showerror("Error", "Please enter a valid angle.")
```

- **cv2.getRotationMatrix2D** and **cv2.warpAffine**: Perform rotation transformation.
- **angle_entry**: Tkinter Entry widget to input rotation angle.

Translate Image Function

```
def translate_image():

    if uploaded_image is None:

        messagebox.showerror("Error", "Please upload an image first.")

        return

    try:

        x = int(x_entry.get())

        y = int(y_entry.get())

        M = np.float32([[1, 0, x], [0, 1, y]])

        (h, w) = uploaded_image.shape[:2]

        result = cv2.warpAffine(uploaded_image, M, (w, h))

        display_image(result, "Translated Image")

    except ValueError:

        messagebox.showerror("Error", "Please enter valid x and y translation values.")

    • np.float32 and cv2.warpAffine: Perform translation transformation.
    • x_entry and y_entry: Tkinter Entry widgets to input translation values.
```

Normalize Image Function

```
def normalize_image():

    if uploaded_image is None:

        messagebox.showerror("Error", "Please upload an image first.")

        return

    result = cv2.normalize(uploaded_image, None, 0, 255, cv2.NORM_MINMAX)

    display_image(result, "Normalized Image")
```

cv2.normalize: Normalizes the image intensity values.

Edge Detection Function

```
def edge_detection():  
    if uploaded_image is None:  
        messagebox.showerror("Error", "Please upload an image first.")  
        return  
    result = cv2.Canny(uploaded_image, 100, 200)  
    display_image(result, "Edge Detected Image")
```

cv2.Canny: Applies Canny edge detection algorithm to the image.

Blur Image Function

```
def blur_image():  
    if uploaded_image is None:  
        messagebox.showerror("Error", "Please upload an image first.")  
        return  
    try:  
        ksize = int(ksize_entry.get())  
        if ksize % 2 == 0:  
            raise ValueError("Kernel size must be odd.")  
        result = cv2.GaussianBlur(uploaded_image, (ksize, ksize), 0)  
        display_image(result, "Blurred Image")  
    except ValueError:  
        messagebox.showerror("Error", "Please enter a valid odd kernel size.")
```

cv2.GaussianBlur: Applies Gaussian blur to the image.

Morphological Operations Function

```
def morphological_operation():  
    if uploaded_image is None:  
        messagebox.showerror("Error", "Please upload an image first.")  
        return  
  
    try:  
        operation = operation_var.get().lower()  
        ksize = int(morph_ksize_entry.get())  
        kernel = np.ones((ksize, ksize), np.uint8)  
  
        if operation == 'erode':  
            result = cv2.erode(uploaded_image, kernel, iterations=1)  
        elif operation == 'dilate':  
            result = cv2.dilate(uploaded_image, kernel, iterations=1)  
        elif operation == 'open':  
            result = cv2.morphologyEx(uploaded_image, cv2.MORPH_OPEN, kernel)  
        elif operation == 'close':  
            result = cv2.morphologyEx(uploaded_image, cv2.MORPH_CLOSE, kernel)  
  
        display_image(result, f"Morphological Operation: {operation}")  
    except ValueError:  
        messagebox.showerror("Error", "Please enter a valid kernel size.")  
  
cv2.erode, cv2.dilate, cv2.morphologyEx: Perform morphological operations (erosion, dilation,  
opening, closing).
```

Display Image Function

```
def display_image(image, title="Image"):
```

```
b,g,r = cv2.split(image)
```

```
img = cv2.merge((r,g,b))
```

```
img = Image.fromarray(img)
```

```
img = ImageTk.PhotoImage(image=img)
```

```
panel.configure(image=img)
```

```
panel.image = img
```

```
panel.update_idletasks()
```

- Converts the OpenCV image format to PIL format for display in Tkinter GUI.
- Updates the image displayed in the GUI panel (`panel`).

Main GUI Setup

```
# Create main window
```

```
root = Tk()
```

```
root.title("Image Processing Tool")
```

```
# Widgets for various image processing functions (buttons, labels, entries)
```

```
# Image display panel
```

```
panel = Label(root)
```

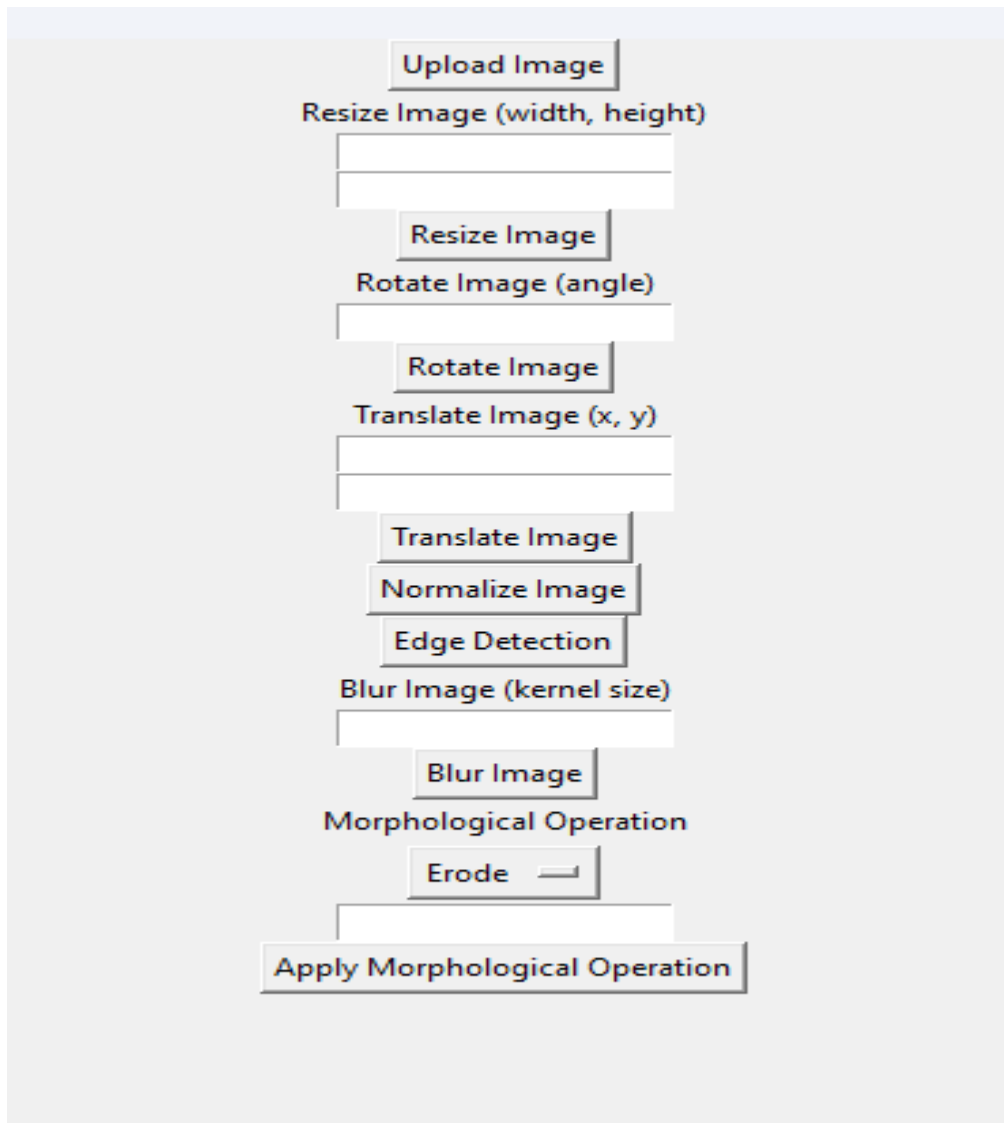
```
panel.pack()
```

```
# Run the GUI
```

```
root.mainloop()
```

- **root**: Main Tkinter window.

- **Buttons, labels, entries:** Various widgets for user interaction.
- **panel:** Tkinter Label widget used for displaying images.



This Python program combines OpenCV for image processing and Tkinter for creating a GUI. It allows users to upload an image and perform various operations such as resizing, rotating, translating, normalizing, edge detection, blurring, and morphological operations on the image.

Each function handles errors gracefully using Tkinter's `messagebox.showerror`. The `display_image` function converts and displays images in the GUI.

TASK-2

AIM : LEARN CNN AND IMPLEMENT IMAGE CLASSIFICATION

OBJECTIVE

The objective of the above program is to develop and train a Convolutional Neural Network (CNN) for image classification using TensorFlow and Keras. Here are the key objectives:

Data Preparation:

- Use `ImageDataGenerator` to load and preprocess images from directories (`train` and `test`).
- Augment training images by rescaling, shearing, zooming, and horizontal flipping to increase the diversity of the training dataset.
- Rescale validation images to maintain consistency in data preprocessing.

Model Architecture:

- Construct a CNN using `Sequential` model from Keras.
- Stack multiple convolutional (`Conv2D`) and pooling (`MaxPooling2D`) layers to extract features from the input images.
- Flatten the output from convolutional layers and pass through fully connected (`Dense`) layers with ReLU activation functions.
- Apply dropout regularization to mitigate overfitting.
- Use a sigmoid activation function in the output layer for binary classification.

Model Compilation:

- Compile the CNN model using the Adam optimizer, which adjusts learning rates during training.
- Use binary cross-entropy loss function suitable for binary classification tasks.
- Track accuracy as a metric to evaluate model performance.

Model Training:

- Train the CNN model using the `fit` method.
- Feed training data (`train_generator`) batch-by-batch to the model for a specified number of epochs.
- Validate the model's performance on the validation data (`validation_generator`) during training.
- Monitor training and validation accuracy and loss metrics to assess model convergence and performance.

Model Evaluation:

- Evaluate the trained model on the validation dataset to obtain validation loss and accuracy.
- Print and analyze these metrics to understand how well the model generalizes to unseen data.

Visualization:

- Plot the training and validation accuracy over epochs to visualize how the model's accuracy improves during training.
- Plot the training and validation loss over epochs to observe how the model's loss decreases, indicating better fit to the data.

Overall, this program aims to build a robust CNN model for image classification tasks, leveraging deep learning techniques and evaluation methodologies to achieve high accuracy and generalization on the validation dataset.

DESCRIPTION

Imports

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
import matplotlib.pyplot as plt
```

- **tensorflow**: TensorFlow library for deep learning.
- **Sequential**: Keras Sequential model to create a linear stack of layers.
- **Conv2D, MaxPooling2D, Flatten, Dense, Dropout**: Different layers used in building the CNN model.
- **ImageDataGenerator**: Generates batches of augmented/normalized data for training and validation.
- **matplotlib.pyplot**: Library for plotting graphs and visualizations.

Constants and Parameters

```
train_dir = 'train'
```

```
validation_dir = 'test'
```

```
img_width, img_height = 150, 150
```

```
input_shape = (img_width, img_height, 3)
```

```
batch_size = 32
```

```
epochs = 20
```

- **train_dir** and **validation_dir**: Directories containing training and validation images.
- **img_width** and **img_height**: Dimensions to which images will be resized.
- **input_shape**: Shape of input images to the CNN (width, height, channels).
- **batch_size**: Number of images in each batch for training.
- **epochs**: Number of times the model will iterate over the entire training dataset.

Data Generators with Augmentation

```
train_datagen = ImageDataGenerator(
```

```
    rescale=1. / 255,
```

```
    shear_range=0.2,
```

```
    zoom_range=0.2,
```

```
    horizontal_flip=True
```

```
)
```

```
validation_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
train_generator = train_datagen.flow_from_directory(
```

```
    train_dir,
```

```
    target_size=(img_width, img_height),
```

```
    batch_size=batch_size,
```

```
    class_mode='binary'
```

```
)
```

```
validation_generator = validation_datagen.flow_from_directory(
```

```
validation_dir,  
target_size=(img_width, img_height),  
batch_size=batch_size,  
class_mode='binary'  
)
```

- **ImageDataGenerator:** Augments and preprocesses images for training and validation.
- **train_datagen:** Augments training images with rescaling, shearing, zooming, and horizontal flipping.
- **validation_datagen:** Only rescales validation images (no augmentation).
- **flow_from_directory:** Generates batches of augmented data from directories of images.
- **class_mode='binary':** Sets the type of classification (binary in this case).

Build CNN Model

```
model = Sequential([  
    Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),  
    MaxPooling2D((2, 2)),  
  
    Conv2D(64, (3, 3), activation='relu'),  
    MaxPooling2D((2, 2)),  
  
    Conv2D(128, (3, 3), activation='relu'),  
    MaxPooling2D((2, 2)),  
  
    Conv2D(128, (3, 3), activation='relu'),  
    MaxPooling2D((2, 2)),
```

```
Flatten(),  
Dense(512, activation='relu'),  
Dropout(0.5),  
Dense(1, activation='sigmoid')  
)
```

```
model.summary()
```

- **Sequential**: Initializes a linear stack of layers.
- **Conv2D**: 2D convolutional layer.
- **MaxPooling2D**: Max pooling operation for spatial data.
- **Flatten**: Flattens the input, effectively turning 2D arrays into 1D array.
- **Dense**: Fully connected layer.
- **Dropout**: Applies dropout regularization to prevent overfitting.
- **activation='relu'** and **activation='sigmoid'**: Rectified Linear Unit (ReLU) and Sigmoid activation functions used in respective layers.

Compile Model

```
model.compile(optimizer='adam',  
              loss='binary_crossentropy',  
              metrics=['accuracy'])
```

- **optimizer='adam'**: Adam optimization algorithm adjusts learning rates during training.
- **loss='binary_crossentropy'**: Binary cross-entropy loss function for binary classification.
- **metrics=['accuracy']**: Evaluates the model based on accuracy during training and testing.

Train Model

```
history = model.fit(  
    train_generator,
```



```
steps_per_epoch=train_generator.samples // batch_size,  
epochs=epochs,  
validation_data=validation_generator,  
validation_steps=validation_generator.samples // batch_size  
)
```

- **model.fit:** Trains the model on data generated batch-by-batch by a Python generator.
- **steps_per_epoch:** Number of batches to yield from the generator per epoch.
- **validation_data:** Data on which to evaluate the loss and any model metrics at the end of each epoch.
- **validation_steps:** Number of batches to yield from the validation generator per epoch.

Evaluate Model

```
score = model.evaluate(validation_generator, verbose=0)  
print('Validation loss:', score[0])  
print('Validation accuracy:', score[1])
```

- **model.evaluate:** Evaluates the model on the validation data.
- **verbose=0:** No output during evaluation.

Plot Training History

```
acc = history.history['accuracy']  
val_acc = history.history['val_accuracy']  
loss = history.history['loss']  
val_loss = history.history['val_loss']
```

```
epochs_range = range(1, epochs + 1)
```

```
plt.figure(figsize=(15, 5))
```

```
plt.subplot(1, 2, 1)
```

```
plt.plot(epochs_range, acc, label='Training Accuracy')
```

```
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
```

```
plt.legend(loc='lower right')
```

```
plt.title('Training and Validation Accuracy')
```

```
plt.subplot(1, 2, 2)
```

```
plt.plot(epochs_range, loss, label='Training Loss')
```

```
plt.plot(epochs_range, val_loss, label='Validation Loss')
```

```
plt.legend(loc='upper right')
```

```
plt.title('Training and Validation Loss')
```

```
plt.show()
```

- **history.history:** Contains training/validation loss and accuracy values.
- **plt.plot:** Plots training/validation accuracy and loss over epochs.
- **plt.subplot:** Divides the figure into subplots for accuracy and loss visualization.
- **plt.legend, plt.title:** Adds legends and titles to the plots.
- **plt.show():** Displays the plotted graphs.

```
Found 1222 images belonging to 2 classes.
Found 1090 images belonging to 2 classes.
C:\Users\HP\PycharmProjects\pythonProject1\venv\lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning:
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-06-21 14:32:03.052705: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags
Model: "sequential"
```

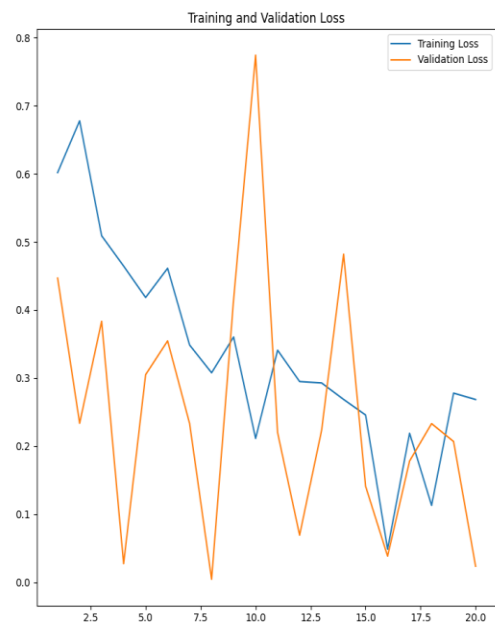
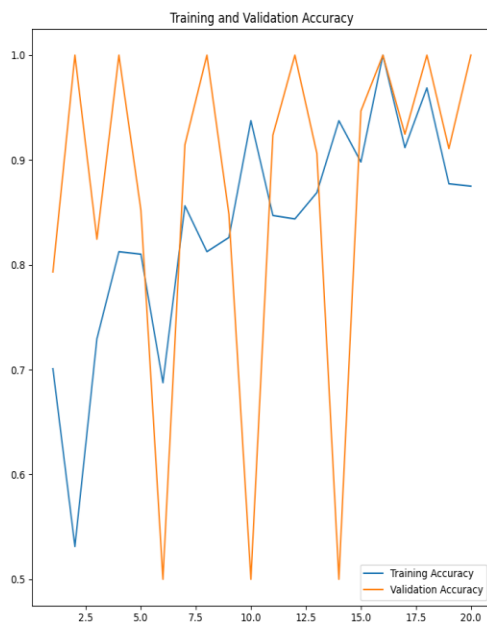
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147,584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	3,211,776
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 1)	513

```
Total params: 3,453,121 (13.17 MB)
Trainable params: 3,453,121 (13.17 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/20
```

```

Epoch 1/20
C:\Users\HP\PycharmProjects\pythonProject1\venv\lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: You are using `GeneratorDatasetAdapter` which is deprecated. Please use `GeneratorDataset` instead.
  self._warn_if_super_not_called()
38/38 ━━━━━━━━━━━ 38s 859ms/step - accuracy: 0.6784 - loss: 0.6227 - val_accuracy: 0.7932 - val_loss: 0.4468
Epoch 2/20
1/38 ━━━━━━━━━━━ 10s 293ms/step - accuracy: 0.5312 - loss: 0.67772024-06-21 14:32:45.105352: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10.3.10.3056.0_x64__qbz5n2kfra8p0\lib\contextlib.py:153: UserWarning: You are using `GeneratorDatasetAdapter` which is deprecated. Please use `GeneratorDataset` instead.
  self.gen.throw(typ, value, traceback)
2024-06-21 14:32:45.211470: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
38/38 ━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5312 - loss: 0.6777 - val_accuracy: 1.0000 - val_loss: 0.2336
Epoch 3/20
38/38 ━━━━━━━━━━━ 23s 553ms/step - accuracy: 0.7042 - loss: 0.5237 - val_accuracy: 0.8244 - val_loss: 0.3834
Epoch 4/20
1/38 ━━━━━━━━━━━ 11s 311ms/step - accuracy: 0.8125 - loss: 0.46442024-06-21 14:33:08.154025: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
2024-06-21 14:33:08.168793: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
38/38 ━━━━━━━━━━━ 0s 674us/step - accuracy: 0.8125 - loss: 0.4644 - val_accuracy: 1.0000 - val_loss: 0.0276
Epoch 5/20
38/38 ━━━━━━━━━━━ 23s 555ms/step - accuracy: 0.7984 - loss: 0.4326 - val_accuracy: 0.8520 - val_loss: 0.3050
Epoch 6/20
1/38 ━━━━━━━━━━━ 9s 264ms/step - accuracy: 0.6875 - loss: 0.46122024-06-21 14:33:30.975165: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
2024-06-21 14:33:30.987009: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
38/38 ━━━━━━━━━━━ 0s 552us/step - accuracy: 0.6875 - loss: 0.4612 - val_accuracy: 0.5000 - val_loss: 0.3546
Epoch 7/20
38/38 ━━━━━━━━━━━ 23s 552ms/step - accuracy: 0.8538 - loss: 0.3620 - val_accuracy: 0.9145 - val_loss: 0.2333
Epoch 8/20
1/38 ━━━━━━━━━━━ 10s 281ms/step - accuracy: 0.8125 - loss: 0.30782024-06-21 14:33:53.917599: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
2024-06-21 14:33:53.930615: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: [[{{node IteratorGetNext}}]]
38/38 ━━━━━━━━━━━ 0s 870us/step - accuracy: 0.8125 - loss: 0.3078 - val_accuracy: 1.0000 - val_loss: 0.0047
Epoch 9/20
38/38 ━━━━━━━━━━━ 23s 562ms/step - accuracy: 0.8310 - loss: 0.3536 - val_accuracy: 0.8483 - val_loss: 0.4154
Epoch 10/20

```



TASK-3

AIM : LEARN THRESHOLD SEGMENTATION TECHNIQUE AND CREATE TOOLS FOR THRESHOLD FUNCTIONS

OBJECTIVE

The objective of the above program is to create an interactive graphical user interface (GUI) application using Python and tkinter that allows users to perform various image thresholding operations on uploaded images. Here are the specific objectives of the program:

Image Upload: Enable users to select an image file from their local system using a file dialog box. Once uploaded, the image is displayed in the GUI.

Thresholding Functions: Implement several image thresholding techniques using Python's PIL (Python Imaging Library):

- **Binary Threshold:** Converts the image to a binary format based on a specified threshold.
- **Binary Inverse Threshold:** Inverts the binary thresholded image.
- **Truncation Threshold:** Thresholds the image by setting values above a specified threshold to the threshold value itself.
- **Threshold to Zero:** Sets pixel values below a threshold to zero.
- **Zero to Threshold:** Sets pixel values above a threshold to zero.

Function Application: Allow users to click on buttons corresponding to each thresholding function to apply the selected operation to the currently displayed image.

Visualization: Display the processed images in real-time within the GUI, updating the image area each time a thresholding function is applied.

User Interaction: Provide a user-friendly interface with buttons for each thresholding operation, enabling users to interactively explore and compare different image processing effects.

Educational and Experimental Use: Serve as a practical tool for users to learn about image thresholding techniques and experiment with different thresholds on their own images.

Integration of Libraries: Demonstrate integration of tkinter for GUI development and PIL for image processing, showcasing how these libraries can be used together to create interactive applications for image manipulation.

Overall, the program aims to facilitate hands-on exploration and understanding of image thresholding methods through an intuitive and accessible graphical interface, enhancing user engagement and learning experience in image processing.

DESCRIPTION

Imports

```
import tkinter as tk
```

```
from tkinter import filedialog
```

```
from PIL import Image, ImageTk
```

- **tkinter:** Standard GUI library for Python.
- **filedialog:** Module within tkinter for handling file dialog boxes.
- **Image, ImageTk:** Classes from the PIL (Python Imaging Library) for image manipulation and display in tkinter.

Thresholding Functions

```
def binary_threshold(image):
```

```
    threshold = 128
```

```
    return image.point(lambda p: p > threshold and 255)
```

```
def binary_inverse_threshold(image):

    threshold = 128

    return image.point(lambda p: p <= threshold and 255)


def trunc_threshold(image):

    threshold = 128

    return image.point(lambda p: min(p, threshold))


def threshold_to_zero(image):

    threshold = 128

    return image.point(lambda p: 0 if p <= threshold else p)
```

```
def zero_to_threshold(image):

    threshold = 128

    return image.point(lambda p: p if p <= threshold else 0)
```

- **Thresholding Functions:** These functions define different image thresholding techniques using lambda functions within PIL's `Image.point` method.
- Each function takes an `image` (PIL Image object) as input and returns a modified image based on specific thresholding logic.

Function to Upload Image

```
def upload_image():

    file_path = filedialog.askopenfilename()

    if file_path:

        global current_image
```

```

current_image = Image.open(file_path)

current_image.thumbnail((300, 300))

photo = ImageTk.PhotoImage(current_image)

label.config(image=photo)

label.image = photo

```

- **upload_image**: Opens a file dialog to select an image file.
- If a file is selected (**file_path** is not empty):
- Opens the image using PIL's **Image.open**.
- Resizes the image to fit within a 300x300 pixel thumbnail.
- Converts the PIL Image to a tkinter-compatible **ImageTk.PhotoImage**.
- Updates the **label** widget in the tkinter GUI to display the uploaded image.

Function to Apply Thresholding

```

def apply_thresholding(function):

    global current_image

    if current_image:

        thresholded_image = function(current_image)

        photo = ImageTk.PhotoImage(thresholded_image)

        label.config(image=photo)

        label.image = photo

```

Tkinter GUI Setup

```

root = tk.Tk()

root.title("Thresholding Functions")

```

- Creates the main tkinter window with the title "Thresholding Functions".

Buttons and Labels

```

upload_button = tk.Button(root, text="Upload Image", command=upload_image)

```



```
upload_button.pack(pady=10)
```

```
label = tk.Label(root)
```

```
label.pack()
```

```
binary_button = tk.Button(root, text="Binary Threshold", command=lambda:  
apply_thresholding(binary_threshold))
```

```
binary_button.pack()
```

```
binary_inverse_button = tk.Button(root, text="Binary Inverse Threshold", command=lambda:  
apply_thresholding(binary_inverse_threshold))
```

```
binary_inverse_button.pack()
```

```
trunc_button = tk.Button(root, text="Trunc Threshold", command=lambda:  
apply_thresholding(trunc_threshold))
```

```
trunc_button.pack()
```

```
threshold_to_zero_button = tk.Button(root, text="Threshold to Zero", command=lambda:  
apply_thresholding(threshold_to_zero))
```

```
threshold_to_zero_button.pack()
```

```
zero_to_threshold_button = tk.Button(root, text="Zero to Threshold", command=lambda:  
apply_thresholding(zero_to_threshold))
```

```
zero_to_threshold_button.pack()
```

- **Buttons:** Create tkinter buttons for various functions.
 - **upload_button:** Triggers `upload_image` function to upload an image.
 - **binary_button, binary_inverse_button, trunc_button, threshold_to_zero_button, zero_to_threshold_button:** Each button applies a specific thresholding function when clicked using `apply_thresholding`.
- **Label:** Creates a tkinter label (`label`) where the uploaded or processed image will be displayed.

Global Variable

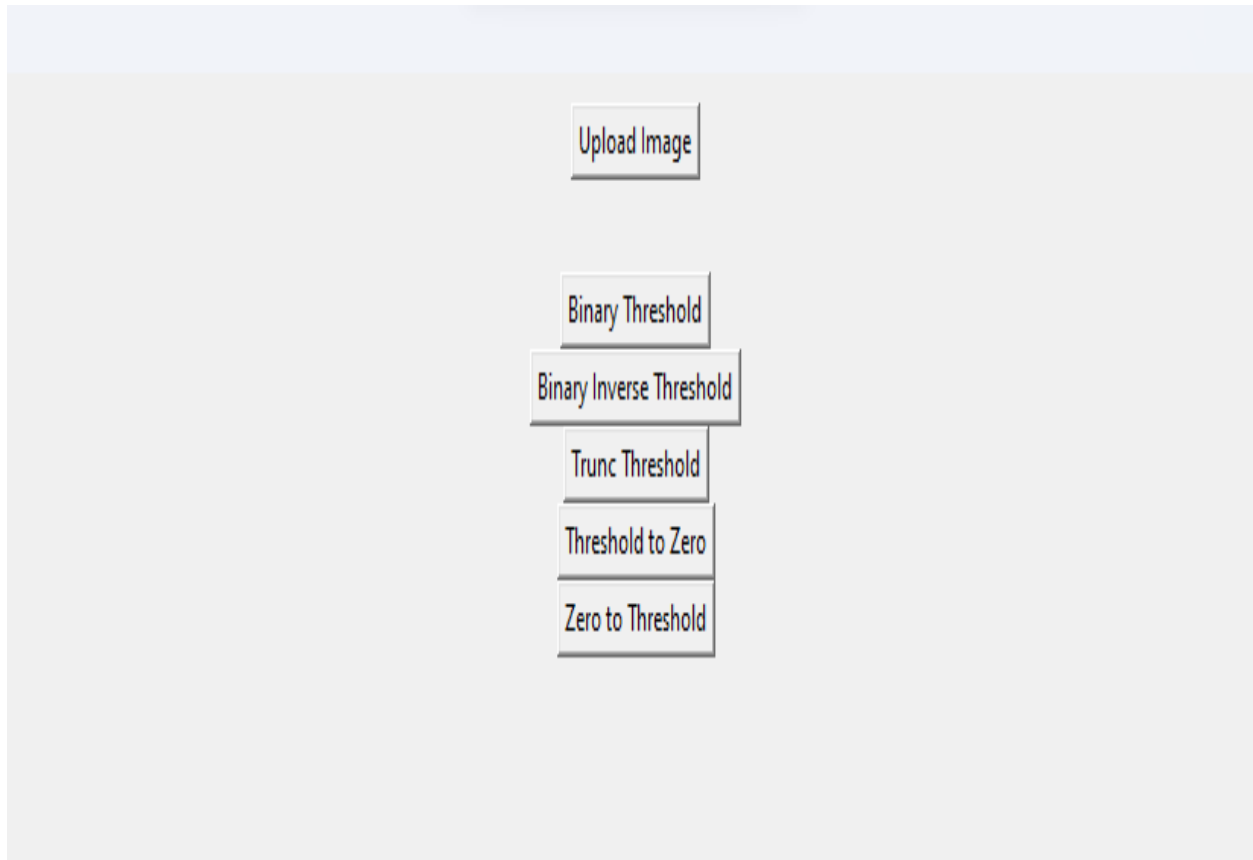
`current_image = None`

current_image: Stores the currently loaded image as a global variable to allow image processing functions (`apply_thresholding`) to access and modify it.

Main Event Loop

`root.mainloop()`

Starts the tkinter main event loop, allowing the GUI to handle user interactions and display updates.



This program creates a graphical user interface (GUI) using tkinter for applying various image thresholding techniques on uploaded images. Users can upload an image, apply different thresholding functions (binary, binary inverse, truncation, etc.), and see the results instantly displayed in the GUI. It demonstrates basic image processing functionalities using PIL and integrates them into a user-friendly interface using tkinter for interactive usage.

TASK-4

AIM : USE THRESHOLD FUNCTION TO EXTRACT CHARACTER FROM AN IMAGE

OBJECTIVE

The objective of the above program is to develop an image processing application using OpenCV (cv2) in Python that performs the following tasks:

Image Loading: Load an image from a specified file path.

Preprocessing:

- Convert the loaded image from color (BGR) to grayscale.
- Apply binary thresholding using Otsu's method (`cv2.THRESH_OTSU`) to create a binary image where characters or objects of interest are highlighted against a uniform background.

Contour Detection:

- Identify contours (outlines) of the objects in the binary image using `cv2.findContours`.
- Extract bounding boxes for each contour to localize and isolate individual characters or objects from the original image.

Character Extraction:

- Utilize the bounding box coordinates to extract regions of interest (characters) from the original image.
- Store each extracted character as a separate image in a list (`characters`).

Visualization:

- Display each extracted character image in separate windows using `cv2.imshow`, enabling visual inspection or further processing.

Example Usage:

- Demonstrate the functionality by applying the above steps to a specific image file ("`C:\\Users\\HP\\Pictures\\dataset\\alpahabet.jpg`" in the provided example).
- Illustrate how to integrate the image processing functions into a cohesive application, facilitating character extraction and visualization for inspection or downstream processing.

User Interaction:

- Provide a command-line interface to initiate the image processing tasks and display the results interactively using OpenCV's window management (`cv2.imshow`, `cv2.waitKey`, `cv2.destroyAllWindows`).

Educational and Practical Application:

- Serve as an educational tool to understand and implement basic image processing techniques like thresholding, contour detection, and object extraction using OpenCV in Python.
- Offer a foundation for more advanced applications such as optical character recognition (OCR), object detection, or image segmentation by showcasing fundamental image processing operations.

DESCRIPTION

Imports

```
import cv2
```

```
# import numpy as np
```

- **cv2**: OpenCV library for computer vision tasks, including image processing and computer vision algorithms.
- **numpy (commented out)**: Typically used for numerical operations, but not explicitly used in this program.

Function extract_characters

```
def extract_characters(image_path):
```

```
    # Load image
```

```
    image = cv2.imread(image_path)
```

```
    # Convert to grayscale
```

```
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
    # Apply binary thresholding
```

```
    _, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)
```

```
    # Find contours
```

```
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,  
cv2.CHAIN_APPROX_SIMPLE)
```

```
    # Extract characters
```

```
    characters = []
```

```
    for contour in contours:
```

```

# Get bounding box coordinates

x, y, w, h = cv2.boundingRect(contour)


# Extract character region from original image

character = image[y:y + h, x:x + w]


# Optionally, you can resize character images here if needed

# character = cv2.resize(character, (desired_width, desired_height))


characters.append(character)


return characters

```

- **extract_characters function:** This function processes an input image to extract individual characters or objects based on thresholding and contour detection.
 - **Load Image:** `cv2.imread(image_path)` loads the image specified by `image_path`.
 - **Convert to Grayscale:** `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` converts the loaded image from BGR (color) to grayscale.
 - **Binary Thresholding:** `cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)` applies binary thresholding to convert the grayscale image to a binary image. `cv2.THRESH_OTSU` automatically determines the optimal threshold value using Otsu's method.
 - **Find Contours:** `cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)` identifies contours in the binary image. `cv2.RETR_EXTERNAL` retrieves only external contours, and `cv2.CHAIN_APPROX_SIMPLE` compresses horizontal, vertical, and diagonal segments and leaves only their end points.
 - **Extract Characters:** Iterates through each contour, calculates its bounding box (`cv2.boundingRect(contour)`), and extracts the corresponding region from

the original image (`image[y:y + h, x:x + w]`). Each extracted character is appended to the `characters` list.

Example Usage (Main Section)

```
if __name__ == "__main__":  
  
    # Path to your image file  
  
    image_path = "C:\\Users\\HP\\Pictures\\dataset\\alpahabet.jpg"  
  
  
    # Extract characters  
  
    extracted_characters = extract_characters(image_path)  
  
  
    # Display or further process extracted characters  
  
    for i, char in enumerate(extracted_characters):  
  
        cv2.imshow(f'Character {i + 1}', char)  
  
  
    cv2.waitKey(0)  
  
    cv2.destroyAllWindows()
```

Main Section: This part of the code demonstrates how to use the `extract_characters` function.

- **Image Path:** Specifies the path to the image file (`image_path`), which contains characters to be extracted.
- **Extract Characters:** Calls `extract_characters(image_path)` to obtain a list of extracted character images.
- **Display or Process Characters:** Iterates through the list of extracted characters (`extracted_characters`), displaying each character image using `cv2.imshow` with a window title (`f'Character {i + 1}'`). Optionally, you can further process or analyze these characters as needed.

- **WaitKey and DestroyAllWindows:** `cv2.waitKey(0)` waits indefinitely until a key is pressed (0 indicates indefinitely), and `cv2.destroyAllWindows()` closes all open windows once processing is complete.



This program demonstrates how to use OpenCV (cv2) to perform image processing tasks such as loading an image, converting it to grayscale, applying binary thresholding, finding contours, and extracting individual characters based on those contours. It provides a practical example of how to automate the extraction of characters or objects from images, which can be useful for applications like optical character recognition (OCR) or object detection in computer vision.

TASK-5

AIM : LEARN AND IMPLEMENT CONTOUR SEGMENTATION TECHNIQUE

OBJECTIVE

The objective of the above program is to demonstrate image segmentation using thresholding and contour detection techniques with OpenCV (cv2) and visualization using Matplotlib (plt). Specifically, the program aims to:

Load an Image: Load a color image

("C:\\Users\\HP\\Pictures\\beagle1.jpg" in this case) using OpenCV and convert it to RGB format for compatibility with Matplotlib.

Convert to Grayscale: Convert the loaded RGB image to grayscale, which simplifies further processing steps.

Apply Binary Thresholding: Apply a binary thresholding technique to the grayscale image. Pixels above a specified intensity threshold (150 in this case) are set to white (255), while pixels below or equal to the threshold are set to black (0).

Find Contours: Detect contours (boundaries) of objects in the binary image using OpenCV's `cv2.findContours` function. This step identifies distinct regions based on intensity differences.

Create a Contour Mask: Create a binary mask based on the detected contours. This mask highlights segmented areas where objects of interest are located against a uniform background.

Apply the Mask: Use the generated mask to segment the original RGB image. This step isolates and extracts the regions of interest based on the contours detected earlier.

Visualize Results: Display the following using Matplotlib:

- The original RGB image (**Original Image**).
- The contour mask (**Contour Mask**), showcasing the areas of interest identified through thresholding and contour detection.
- The segmented image (**Segmented Image**), where the original image is masked to highlight the identified regions of interest.

Educational Purpose: Serve as an educational tool to illustrate fundamental image processing techniques such as thresholding, contour detection, and masking using OpenCV and Matplotlib.

Demonstration: Provide a clear demonstration of how these techniques can be implemented in Python to segment objects within an image, facilitating further analysis or processing tasks in computer vision applications.

DESCRIPTION

Imports

```
import cv2
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

- **cv2:** OpenCV library for computer vision tasks, including image processing and computer vision algorithms.

- **numpy (np)**: Library for numerical operations in Python, used here for array operations.
- **matplotlib.pyplot (plt)**: Library for plotting and visualizing data in Python, used here to display images.

Load Image

```
image = cv2.imread("C:\\Users\\HP\\Pictures\\beagle1.jpg")
```

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert to RGB (matplotlib expects RGB)
```

- **cv2.imread**: Loads the image from the specified file path ("C:\\Users\\HP\\Pictures\\beagle1.jpg" in this case).
- **cv2.cvtColor**: Converts the image from BGR (default in OpenCV) to RGB format, which is required by Matplotlib for proper display.

Convert to Grayscale

```
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

cv2.cvtColor: Converts the RGB image to grayscale. Grayscale images have a single channel representing intensity, which simplifies further processing.

Apply Binary Thresholding

```
_, binary = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
```

cv2.threshold: Applies binary thresholding to the grayscale image (**gray**).

- **gray**: Input grayscale image.
- **150**: Threshold value. Pixels with intensity greater than 150 are set to 255 (white), and pixels with intensity less than or equal to 150 are set to 0 (black).
- **255**: Maximum pixel value after applying threshold.
- **cv2.THRESH_BINARY**: Thresholding type that converts pixels above the threshold to a maximum value (255 in this case).

Find Contours

```
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

cv2.findContours: Detects contours (boundaries) in the binary image (**binary**).

- **binary**: Input binary image obtained from thresholding.

- `cv2.RETR_EXTERNAL`: Retrieves only the external contours.
- `cv2.CHAIN_APPROX_SIMPLE`: Compresses horizontal, vertical, and diagonal segments, keeping only their endpoints.

Create Contour Mask

```
mask = np.zeros_like(gray)
```

```
cv2.drawContours(mask, contours, -1, (255, 255, 255), thickness=cv2.FILLED)
```

- **`np.zeros_like`**: Creates an array of zeros with the same shape and type as `gray`.
- **`cv2.drawContours`**: Draws contours (`contours`) onto the mask (`mask`).
- `-1`: Draw all contours found.
- `(255, 255, 255)`: Color of the contours (white).
- `thickness=cv2.FILLED`: Fills the contours with white color, creating a mask of the segmented areas.

Apply Mask to Original Image

```
segmented_image = cv2.bitwise_and(image, image, mask=mask)
```

`cv2.bitwise_and`: Applies bitwise AND operation between `image` and `mask`.

- `image`: Original RGB image.
- `mask`: Binary mask where segmented areas are white (255), and background is black (0).
- Result (`segmented_image`) retains only the parts of `image` where `mask` is non-zero, effectively segmenting the image based on contours.

Display Results using Matplotlib

```
plt.figure(figsize=(12, 6))
```

```
plt.subplot(1, 3, 1)
```

```
plt.imshow(image)
```

```
plt.title('Original Image')
```

```
plt.axis('off')
```

```
plt.subplot(1, 3, 2)

plt.imshow(mask, cmap='gray')

plt.title('Contour Mask')

plt.axis('off')
```

```
plt.subplot(1, 3, 3)

plt.imshow(segmented_image)

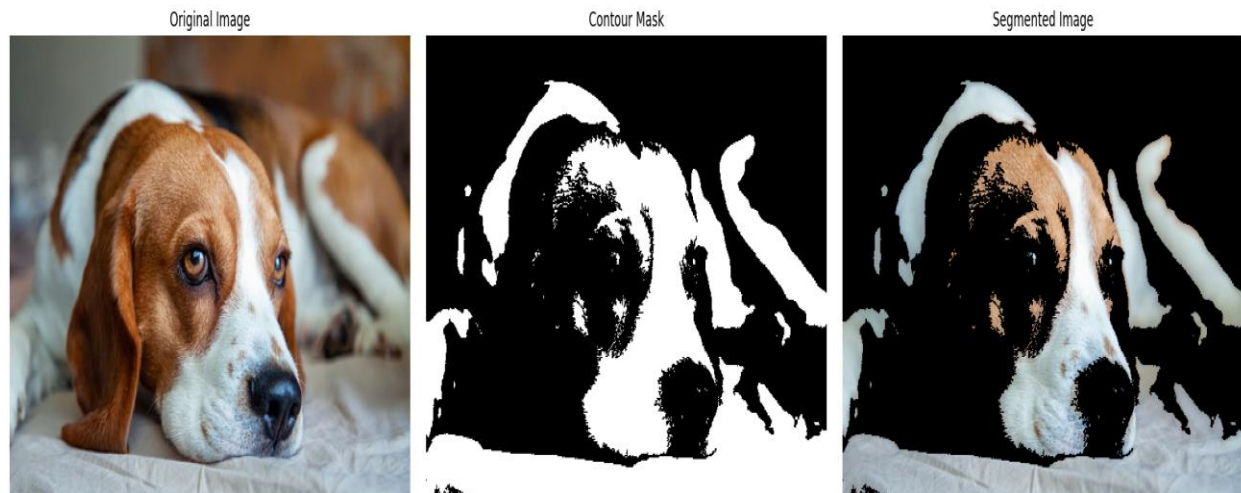
plt.title('Segmented Image')

plt.axis('off')
```

```
plt.tight_layout()

plt.show()
```

- **plt.figure:** Creates a new figure for plotting with specified size (`figsize=(12, 6)`).
- **plt.subplot:** Specifies a grid of subplots (1 row, 3 columns).
- **plt.imshow:** Displays images within each subplot.
- **plt.title:** Sets titles for each subplot.
- **plt.axis('off'):** Removes axis ticks and labels for cleaner visualization.
- **plt.tight_layout():** Adjusts subplot parameters for better layout.
- **plt.show():** Displays the entire figure with all subplots.



This program demonstrates a comprehensive image segmentation process using thresholding and contour detection with OpenCV. It loads an image, converts it to grayscale, applies binary thresholding to create a mask, finds contours, draws contours on the mask, and uses the mask to segment the original image. Finally, it visualizes the original image, contour mask, and segmented image side by side using Matplotlib for clear inspection and analysis.

TASK-6

AIM : LEARN AND IMPLEMENT WATER_SHED SEGMENTATION TECHNIQUE

OBJECTIVE

The objective of the provided program is to perform coin segmentation from an input image using computer vision techniques, specifically leveraging the watershed algorithm. Here's a breakdown of the objectives achieved by the program:

Image Loading and Preprocessing:

- Load an image of coins and convert it to grayscale.
- Apply thresholding to create a binary image for further processing.
- Perform morphological operations to clean up noise in the binary image.

Segmentation Preparation:

- Identify and separate the sure background and sure foreground areas.
- Determine the unknown region where the algorithm needs to further delineate object boundaries.

Watershed Algorithm Application:

- Use the watershed algorithm to segment the image into distinct regions corresponding to individual coins.
- Mark the segmented regions with different labels to differentiate them.

Contours Extraction and Visualization:

- Extract contours of segmented regions to precisely outline each coin.
- Overlay the detected contours onto the original image to visually highlight the boundaries of each coin.

Visualization:

- Display intermediate results and final segmented image with marked contours using matplotlib.

In essence, the program aims to automate the process of identifying and delineating individual coins from an input image using advanced image processing techniques, facilitating tasks such as coin counting, measurement, or further analysis in various applications.

DESCRIPTION

Importing Libraries

```
import cv2
import numpy as np
from IPython.display import Image, display
from matplotlib import pyplot as plt
```

- **cv2**: OpenCV library for computer vision tasks.
- **numpy**: Library for numerical operations, used for handling arrays/matrices.
- **IPython.display**: Used to display images in Jupyter notebooks.
- **matplotlib.pyplot**: Plotting library for visualization

imshow Function

```
def imshow(img, ax=None):
    if ax is None:
        ret, encoded = cv2.imencode(".jpg", img)
        display(Image(encoded))
    else:
        ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        ax.axis('off')
```

- **imshow**: A function to display images either in a standalone window (if **ax=None**) or within a matplotlib axes (**ax**).
- **cv2.imencode**: Encodes the image to JPEG format.
- **display**: Displays the encoded image.
- **ax.imshow**: Displays the image in matplotlib.
- **cv2.cvtColor**: Converts image colorspace from BGR to RGB.

Image Loading

```
img = cv2.imread("C:\\Users\\HP\\Pictures\\coin.jpg")
```

cv2.imread: Loads an image from the specified file path.

Image Processing Steps

```
#Convert to Grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

cv2.COLOR_BGR2GRAY: Converts BGR (color) image to grayscale.

```
#Threshold Processing
ret, bin_img = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
```

- **cv2.threshold**: Applies a threshold to convert a grayscale image to a binary image.
- **cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU**: Otsu's thresholding method for automatic threshold selection and inverse binary thresholding.

```
#Noise Removal
```

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
```

```
bin_img = cv2.morphologyEx(bin_img, cv2.MORPH_OPEN, kernel, iterations=2)
```

- **cv2.getStructuringElement:** Creates a structuring element (kernel) for morphological operations.
- **cv2.morphologyEx:** Applies morphological transformations (opening) to remove noise.

```
#Matplotlib Subplots
```

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))
```

- **plt.subplots:** Creates a figure and a set of subplots.
- **nrows=2, ncols=2:** Creates a 2x2 grid of subplots.

Further Image Processing Steps

```
#Sure Background
```

```
sure_bg = cv2.dilate(bin_img, kernel, iterations=3)
```

- **cv2.dilate:** Dilates the image to enhance the foreground objects.

```
#Distance Transform
```

```
dist = cv2.distanceTransform(bin_img, cv2.DIST_L2, 5)
```

- **cv2.distanceTransform:** Calculates the distance to the closest zero pixel for each pixel in the image.

```
#Sure Foreground
```

```
ret, sure_fg = cv2.threshold(dist, 0.5 * dist.max(), 255, cv2.THRESH_BINARY)
```

- **cv2.threshold:** Thresholds the distance transform image to obtain the sure foreground.

```
#Unknown Area
```

```
unknown = cv2.subtract(sure_bg, sure_fg)
```

cv2.subtract: Subtracts the sure foreground from sure background to get the unknown region.

Marker Labelling and Watershed Algorithm

```
ret, markers = cv2.connectedComponents(sure_fg)
```

```
markers += 1
```

```
markers[unknown == 255] = 0
```

```
markers = cv2.watershed(img, markers)
```

- **cv2.connectedComponents:** Labels the connected components in the sure foreground.
- **cv2.watershed:** Applies the watershed algorithm to segment the image.

Contour Extraction and Drawing

```
labels = np.unique(markers)
```

```
coins = []
```

```
for label in labels[2:]:
```

```
    target = np.where(markers == label, 255, 0).astype(np.uint8)
```

```
    contours, hierarchy = cv2.findContours(target, cv2.RETR_EXTERNAL,  
cv2.CHAIN_APPROX_SIMPLE)
```

```
    coins.append(contours[0])
```

```
img = cv2.drawContours(img, coins, -1, color=(0, 23, 223), thickness=2)
```

```
imshow(img)
```

- **np.unique:** Finds unique elements in the array.
- **cv2.findContours:** Finds contours in the binary image.
- **cv2.drawContours:** Draws contours on the original image.

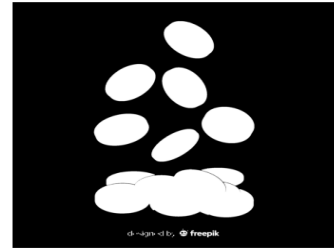
File info



Sure Background



Distance Transform



Sure Foreground



Unknown





This program performs image segmentation of coins using the watershed algorithm:

- Loads an image and converts it to grayscale.
- Applies thresholding and morphological operations to prepare the image.
- Calculates the sure background, sure foreground, and unknown regions.
- Uses watershed algorithm for segmentation and draws contours around detected coins.

Each step involves using OpenCV functions for image processing and matplotlib for visualization.

TASK-7

AIM : LEARN AND IMAPLEMENT REGION_BASE SEGMENTATION TECHNIQUE

OBJECTVE

The objective of the provided Python program is to implement and demonstrate the basic functionality of the region growing algorithm for grayscale image segmentation. Here's a breakdown of the objectives achieved by the program:

Image Loading:

- Read a grayscale image (beagle1.jpg) using OpenCV (cv2.imread).

Region Growing Algorithm Implementation:

- Implement a basic region growing algorithm (region_growing function) to segment the grayscale image based on intensity similarity starting from a user-defined seed point.

Segmentation Process:

- Start segmentation from a manually selected seed point ((100, 100) in this example).
- Grow the segmented region by iteratively adding neighboring pixels that have intensity values similar to the seed point, controlled by a predefined intensity difference threshold.

Output Visualization:

- Display the original grayscale image ('Original Image') alongside the segmented image ('Segmented Image') using OpenCV functions (cv2.imshow, cv2.waitKey, cv2.destroyAllWindows).

Demonstration and Understanding:

- Provide a visual demonstration of how region growing can be used for image segmentation, where pixels with similar intensity values to the seed point are grouped together into a coherent region.

Educational and Experimental Purposes:

- Serve as an educational tool to understand the principles of region growing algorithm in image processing.

- Allow experimentation with different seed points and threshold values to observe their effects on segmentation results.

In essence, the program aims to introduce and showcase the basic mechanics of region growing algorithm for grayscale image segmentation, encouraging exploration and further development in the field of computer vision and image analysis applications.

DESCRIPTION

Imports and Function Definition

```
import numpy as np
```

```
import cv2
```

- **numpy**: Library for numerical operations, used for handling arrays/matrices.
- **cv2**: OpenCV library for computer vision tasks, including image processing and computer vision algorithms.

```
def region_growing(image, seed):
```

```
    """
```

Basic region growing algorithm for grayscale images.

Parameters:

- image: Input grayscale image (2D numpy array).
- seed: Seed point (tuple of (row, col)).

Returns:

- segmented_image: Binary mask indicating segmented region.

```
    """
```

- **region_growing function**: Implements a basic region growing algorithm for grayscale images.
- **Parameters**:

- **image:** Input grayscale image represented as a 2D numpy array.
- **seed:** Seed point from which the region growing starts, specified as a tuple (row, col).
- **Returns:**
- **segmented_image:** A binary mask (2D numpy array) indicating the segmented region where pixels belonging to the region are set to 255 (white) and others to 0 (black).

Initializing Parameters

Dimensions of input image

rows, cols = image.shape

Threshold for region growing (example: intensity difference)

threshold_diff = 20

Output segmented image (binary mask)

segmented_image = np.zeros_like(image, dtype=np.uint8)

- **image.shape:** Retrieves the dimensions (rows and columns) of the input image.
- **threshold_diff:** Defines a threshold for the intensity difference between the seed point and its neighbors for region growing.
- **segmented_image:** Initializes an output image of the same dimensions as **image**, filled with zeros (`dtype=np.uint8` ensures it's an 8-bit unsigned integer array).

Region Growing Algorithm

Queue for region growing

queue = []

queue.append(seed)

Seed point intensity value

```

seed_intensity = image[seed]

# Process until the queue is empty
while queue:

    # Get current seed point
    current_point = queue.pop(0)

    # Check 8-connected neighbors
    for i in range(-1, 2):
        for j in range(-1, 2):

            # Skip out-of-bound pixels
            if (current_point[0] + i < 0 or current_point[0] + i >= rows or
                current_point[1] + j < 0 or current_point[1] + j >= cols):
                continue

            # Check if neighbor pixel is unvisited and within threshold
            if segmented_image[current_point[0] + i, current_point[1] + j] == 0:
                neighbor_intensity = image[current_point[0] + i, current_point[1] + j]
                intensity_diff = np.abs(int(seed_intensity) - int(neighbor_intensity))

                if intensity_diff < threshold_diff:
                    segmented_image[current_point[0] + i, current_point[1] + j] = 255
                    queue.append((current_point[0] + i, current_point[1] + j))

```

- **Queue Initialization:** Starts with a queue containing the seed point.
- **Seed Intensity:** Retrieves the intensity value of the seed point from the input image.

- **While Loop:** Executes the region growing algorithm until the queue is empty.
- **Neighborhood Check:** Iterates over 8-connected neighbors of the current seed point.
- **Boundary Check:** Skips pixels that are out of image bounds.
- **Unvisited and Threshold Check:** Checks if the neighbor pixel hasn't been visited (`segmented_image == 0`) and if the intensity difference between the neighbor and seed points is within `threshold_diff`.
- **Segmentation:** Marks the neighbor pixel as part of the segmented region (`segmented_image[current_point[0] + i, current_point[1] + j] = 255`) and adds it to the queue for further processing.

Example Usage

Example usage:

```
if __name__ == "__main__":
```

```
    # Read an image (grayscale)
```

```
    image = cv2.imread("C:\\Users\\HP\\Pictures\\beagle1.jpg", cv2.IMREAD_GRAYSCALE)
```

```
    # Select a seed point (manually or using some algorithm)
```

```
    seed_point = (100, 100) # Example seed point
```

```
    # Perform region growing segmentation
```

```
    segmented_image = region_growing(image, seed_point)
```

```
    # Display the original image and segmented image
```

```
    cv2.imshow('Original Image', image)
```


```
    cv2.imshow('Segmented Image', segmented_image)
```

```
    cv2.waitKey(0)
```

```
    cv2.destroyAllWindows()
```

- **Reading Image:** Uses OpenCV (`cv2.imread`) to read an input grayscale image (beagle1.jpg in this case).
- **Seed Point Selection:** Manually selects a seed point (100, 100) in the image for region growing.
- **Segmentation:** Calls `region_growing` function to segment the region around the seed point.
- **Display:** Shows the original grayscale image and the segmented result using OpenCV's `cv2.imshow`, `cv2.waitKey`, and `cv2.destroyAllWindows` functions.



 Original Image





This program demonstrates a basic implementation of the region growing algorithm for grayscale images, where pixels are grouped into regions based on intensity similarity to a seed point. The segmented image highlights the connected region starting from the seed point, providing a simple yet effective example of image segmentation.

TASK-8

AIM : LEARN AND IMPLEMENT CANNY_EDGE_DETECTION SEGMENTATION TECHNIQUE

OBJECTIVE

Objective:

The objective of this program is to perform Canny edge detection on a given input image (beagle1.jpg). The steps involved include loading the image, resizing it for display purposes, converting it to grayscale to simplify processing, applying Gaussian blur to reduce noise, and finally applying the Canny edge detection algorithm to detect edges in the image. The program then displays both the original image and the detected edges to visually demonstrate the effect of edge detection. This serves as a fundamental example of how to apply edge detection techniques using OpenCV in Python for image processing tasks.

DESCRIPTION

Imports and Image Loading

```
import cv2
```

```
import numpy as np
```

- **cv2**: OpenCV library for computer vision tasks.
- **numpy (np)**: Library for numerical operations in Python.

```
# Load an image
```

```
image = cv2.imread("C:\\Users\\HP\\Pictures\\beagle1.jpg")
```

- **cv2.imread**: Loads an image from the specified file path ("C:\\Users\\HP\\Pictures\\beagle1.jpg" in this case).
- **image**: Stores the loaded image data as a numpy array.

Error Handling

```
# Check if the image was loaded properly  
if image is None:
```

```
print("Error: Could not load image.")
exit()
```

Error Handling: Checks if the image loading was successful. If `image` is `None`, it prints an error message and exits the program.

Image Resizing

```
# Resize the image for display (if needed)
```

```
max_display_width = 800
```

```
max_display_height = 600
```

```
# Get the original image dimensions
```

```
height, width = image.shape[:2]
```

```
# Calculate the scaling factor to fit within the display dimensions
```

```
scale = min(max_display_width / width, max_display_height / height)
```

```
# Resize the image
```

```
resized_image = cv2.resize(image, None, fx=scale, fy=scale)
```

- **Image Resizing:** Ensures the image fits within a maximum display width and height (`max_display_width` and `max_display_height`).
- **`image.shape`:** Returns the dimensions of the original image (`height` and `width`).
- **`scale`:** Calculates the scaling factor to resize the image proportionally.
- **`cv2.resize`:** Resizes the image using the calculated scaling factor (`fx` and `fy`).

Grayscale Conversion and Gaussian Blur

```
# Convert the resized image to grayscale
```

```
gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
```

```
# Apply Gaussian blur to the grayscale image
```

```
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

Canny Edge Detection

```
# Apply Canny edge detection
```

```
edges = cv2.Canny(blurred, 50, 150)
```

- **cv2.Canny:** Performs Canny edge detection on the blurred image (blurred).
 - **50:** Lower threshold for the hysteresis procedure of the Canny edge detector.
 - **150:** Upper threshold for the hysteresis procedure of the Canny edge detector.
 - These thresholds control which edges are detected. Typically, a ratio of 1:3 to 1:2 between the lower and upper threshold is recommended.

Displaying Images

Display the original image and the edges detected

```
cv2.imshow('Original Image', resized_image)
```

```
cv2.imshow('Canny Edges', edges)
```

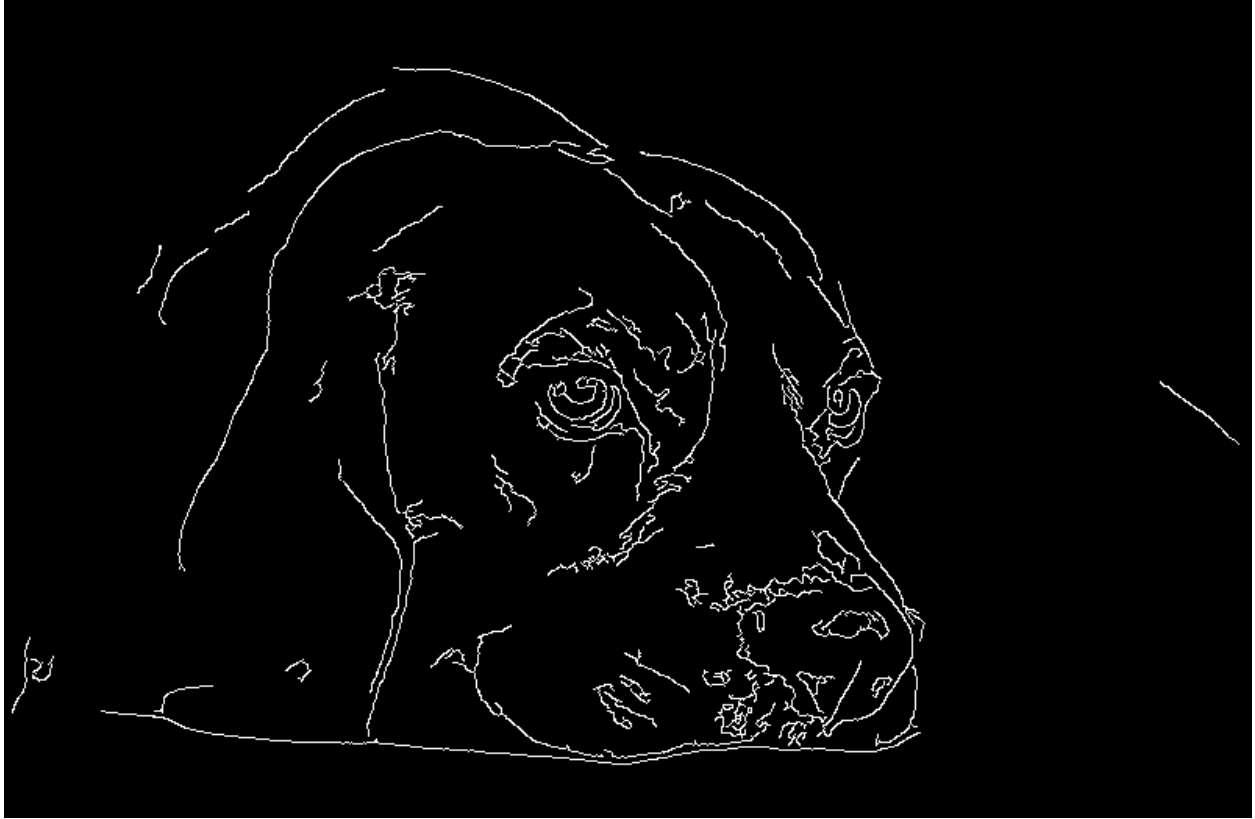
Wait for a key press and close all windows

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

- **cv2.imshow:** Displays images in separate windows with specified window names ('Original Image' and 'Canny Edges').
- **cv2.waitKey(0):** Waits indefinitely for a key press. Once a key is pressed, it continues execution.
- **cv2.destroyAllWindows():** Closes all OpenCV windows.





This program loads an image, resizes it for display, converts it to grayscale, applies Gaussian blur, performs Canny edge detection, and then displays both the original image and the detected edges using OpenCV. It's a typical workflow for basic edge detection in computer vision applications.

TASK-9

AIM : LEARN SIFT FEATURE AND FIND KEYPOIN MATCHES BETWEEN IMAGES

OBJECTIVE

The objective of this program is to demonstrate key aspects of feature detection and matching using the Scale-Invariant Feature Transform (SIFT) algorithm between two selected images. The program achieves the following:

Image Selection: Allows the user to interactively select two images from their file system.

Image Processing:

- Reads the selected images.
- Converts the images to grayscale to simplify feature detection.

Feature Detection:

- Applies the SIFT algorithm to detect keypoints (distinctive features) in both grayscale images.
- Computes descriptors (feature vectors) associated with each keypoint.

Feature Matching:

- Utilizes the Fast Library for Approximate Nearest Neighbors (FLANN) to match keypoints between the two images based on their descriptors.
- Filters and selects "good matches" using Lowe's ratio test to ensure robustness against mismatches.

Visualization:

- Draws and displays the matches between keypoints on the original images using OpenCV's `drawMatches` function.
- Provides a visual representation of how well keypoints from one image correspond to those in the other.

Analysis:

- Constructs histograms of keypoint sizes to analyze and compare the distribution of feature sizes between the two images.
- Offers insights into the scale and frequency of detected features, aiding in understanding the content and quality of feature detection.

Overall, this program serves as an educational tool to illustrate the capabilities of SIFT for feature-based image alignment and comparison. It emphasizes the steps involved in feature detection, matching, and analysis, showcasing their importance in various computer vision applications such as image stitching, object recognition, and more.

DESCRIPTION

Imports

```
import cv2
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from tkinter import Tk, filedialog
```

- **cv2**: OpenCV library for computer vision tasks.
- **numpy (np)**: Library for numerical operations in Python.
- **matplotlib.pyplot (plt)**: Library for creating plots and visualizations.
- **tkinter**: Library for creating GUI applications in Python. Used here for file dialog to select images.

upload_image Function

```
def upload_image():
```

```
    root = Tk()
```

```
    root.withdraw() # Close the root window
```

```
    file_path = filedialog.askopenfilename(title="Select an Image",
```

```
                                           filetypes=[("Image files", "*.jpg;*.jpeg;*.png;*.bmp")])
```

```
    return file_path
```

upload_image: Function that opens a file dialog for selecting an image file.

- **Tk()**: Creates a Tkinter root window (hidden with `root.withdraw()`).
- **filedialog.askopenfilename**: Displays a file dialog to select an image file (*.jpg, *.jpeg, *.png, *.bmp).
- **return file_path**: Returns the selected file path.

process_images Function

```
def process_images(img1_path, img2_path):
```

```
    # Read the images
```

```
    img1 = cv2.imread(img1_path)
```

```
    img2 = cv2.imread(img2_path)
```

```
# Convert images to grayscale
```

```
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
```

```
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
```

- **process_images:** Function that processes two input images.
 - **cv2.imread:** Reads images from file paths (`img1_path` and `img2_path`).
 - **cv2.cvtColor:** Converts BGR images (`img1` and `img2`) to grayscale (`gray1` and `gray2`).

SIFT Feature Detection and Matching

```
# Initialize SIFT detector
```

```
sift = cv2.SIFT_create()
```

```
# Detect SIFT features and compute descriptors
```

```
keypoints_1, descriptors_1 = sift.detectAndCompute(gray1, None)
```

```
keypoints_2, descriptors_2 = sift.detectAndCompute(gray2, None)
```

SIFT (Scale-Invariant Feature Transform):

- **cv2.SIFT_create():** Initializes the SIFT detector.
- **sift.detectAndCompute:** Detects keypoints (`keypoints_1` and `keypoints_2`) and computes descriptors (`descriptors_1` and `descriptors_2`) for the grayscale images (`gray1` and `gray2`).

FLANN Matcher Setup and Matching

```
# FLANN parameters
```

```
FLANN_INDEX_KDTREE = 1
```

```
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
```

```
search_params = dict(checks=50)
```

```
flann = cv2.FlannBasedMatcher(index_params, search_params)
```

```
# Match descriptors
```

```
matches = flann.knnMatch(descriptors_1, descriptors_2, k=2)
```

```
# Store all the good matches as per Lowe's ratio test
good_matches = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
        good_matches.append(m)
```

FLANN (Fast Library for Approximate Nearest Neighbors) Matcher:

- **cv2.FlannBasedMatcher:** Initializes the FLANN-based matcher with specified index parameters (`index_params`) and search parameters (`search_params`).
- **flann.knnMatch:** Matches descriptors (`descriptors_1` from `gray1` and `descriptors_2` from `gray2`) using the FLANN matcher.
- **Lowe's Ratio Test:** Filters matches based on a distance ratio (`0.7`), selecting only `good_matches`.

Drawing and Displaying Matches

```
# Draw matches
img_matches = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2, good_matches,
None,
                             flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
```

```
# Display the matched features
plt.figure(figsize=(20, 10))
plt.imshow(cv2.cvtColor(img_matches, cv2.COLOR_BGR2RGB))
plt.title('Feature Matching')
plt.show()
```

- **cv2.drawMatches:** Draws keypoints and their matches (`good_matches`) between `img1` and `img2`.
- **matplotlib.pyplot:** Displays the drawn matches using `plt.imshow`.

Histogram of Keypoint Sizes

```
# Compute histograms of keypoint sizes
kp1_sizes = [kp.size for kp in keypoints_1]
kp2_sizes = [kp.size for kp in keypoints_2]

plt.figure(figsize=(12, 6))
plt.hist(kp1_sizes, bins=50, alpha=0.5, label='Image 1')
```

```
plt.hist(kp2_sizes, bins=50, alpha=0.5, label='Image 2')
plt.legend(loc='upper right')
plt.title('Histogram of SIFT Keypoint Sizes')
plt.xlabel('Keypoint Size')
plt.ylabel('Frequency')
plt.show()
```

Histogram of Keypoint Sizes:

- Computes and plots histograms (`kp1_sizes` and `kp2_sizes`) of SIFT keypoints' sizes for `gray1` and `gray2`.
- Provides insights into the distribution and frequency of keypoints' sizes in both images.

Main Execution

```
if __name__ == "__main__":
    print("Please select the first image.")
    img1_path = upload_image()

    if not img1_path:
        print("No image selected. Exiting.")
        exit()

    print("Please select the second image.")
    img2_path = upload_image()

    if not img2_path:
        print("No image selected. Exiting.")
        exit()

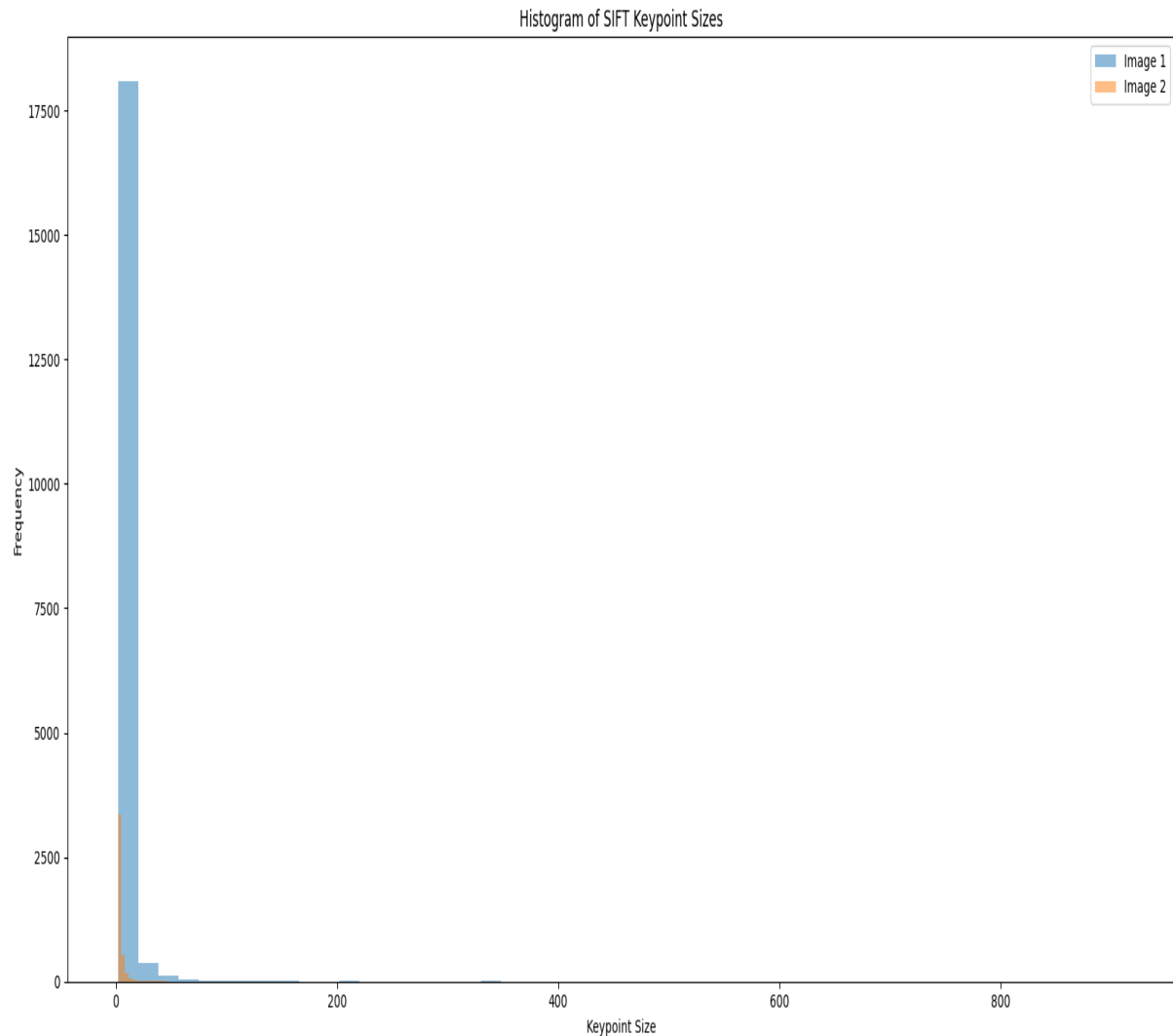
    process_images(img1_path, img2_path)
```

Main Execution: Controls the flow of the program.

- Prompts the user to select two images (`img1` and `img2`) using `upload_image`.
- Calls `process_images` to perform SIFT feature detection, matching, and visualization on the selected images.

Feature Matching





This program utilizes OpenCV and Matplotlib to perform SIFT feature detection and matching between two selected images. It demonstrates the steps of loading images, converting them to grayscale, detecting SIFT keypoints, matching keypoints between images, drawing and displaying matches, and visualizing histograms of SIFT keypoint sizes. The program is structured to allow interactive selection of images and provides visual and statistical insights into feature matching using SIFT.

TASK-10

AIM : OBJECT DETECTION USING ALREADY EXISTING MODEL LIKE ALEXNET

OBJECTIVE

The objective of the program is to create a graphical user interface (GUI) application that allows users to:

Upload an Image: Users can select an image file from their local system using a file dialog.

Classify the Image: Once an image is selected, the program preprocesses the image and uses a pre-trained AlexNet model to classify it into one of the classes from the ImageNet dataset.

Display Classification Result: The program displays the uploaded image along with the predicted label and confidence score in a separate window.

Error Handling: The program includes error handling to manage potential issues during image processing or classification, providing user-friendly error messages when necessary.

Overall, the program aims to demonstrate the integration of deep learning-based image classification with a simple and interactive GUI using Python's Tkinter library. This allows users to experience real-time image classification results directly from their desktop environment.

DESCRIPTION

Imports and Initialization

```
import torch

import torchvision.transforms as transforms

import torchvision.models as models

from PIL import Image, ImageTk

import requests

from io import BytesIO

import tkinter as tk

from tkinter import filedialog, messagebox
```

Imports: Import necessary libraries and modules. Notably:

- `torch` and related modules (`transforms`, `models`): For deep learning operations.
- `PIL.Image` and `ImageTk`: For handling images in PIL (Python Imaging Library) format and Tkinter-compatible images.
- `requests` and `io.BytesIO`: For fetching data over HTTP and handling byte streams.
- `tkinter` and specific components (`filedialog`, `messagebox`): For building the graphical user interface (GUI).

Load Pre-trained Model and Labels

```
model = models.alexnet(pretrained=True)
```

```
model.eval()
```

```
preprocess = 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]),
])
```

```
labels_url = "https://raw.githubusercontent.com/anishathalye/imagenet-simple-
labels/master/imagenet-simple-labels.json"
```

```
labels = requests.get(labels_url).json()
```

- **Model Loading:** Loads a pre-trained AlexNet model from torchvision's `models` module and sets it to evaluation mode (`eval()`).
- **Image Preprocessing:** Defines a series of transformations (`preprocess`) to be applied to images before feeding them into the model for classification.
- **Labels:** Fetches ImageNet class labels from a JSON file hosted online.

Function Definitions

```
classify_and_display(image_path)
```

```
def classify_and_display(image_path):
```

try:

```
image = Image.open(image_path)
```

```
image_tensor = preprocess(image)
```

```
image_tensor = image_tensor.unsqueeze(0)
```

```
with torch.no_grad():
```

```
    output = model(image_tensor)
```

```
    probabilities = torch.nn.functional.softmax(output[0], dim=0)
```

```
_, idx = torch.max(probabilities, 0)
```

```
predicted_label = labels[idx.item()]
```

```
confidence = probabilities[idx.item()].item()
```

```
display_image_with_prediction(image, predicted_label, confidence)
```

except Exception as e:

```
    messagebox.showerror("Error", f"Error processing image: {str(e)}")
```

- **Image Classification:** Loads an image, preprocesses it, passes it through the model to obtain predictions, and then calls `display_image_with_prediction()` to show the results.
- **Error Handling:** Catches and displays errors that may occur during image processing.

Display_image_D with_prediction(image, predicted_label, confidence)

```
def display_image_with_prediction(image, predicted_label, confidence):  
  
    window = tk.Toplevel(root)  
  
    window.title("Image Classification Result")  
  
  
    image = ImageTk.PhotoImage(image)  
  
    label_image = tk.Label(window, image=image)  
  
    label_image.image = image  
  
    label_image.pack()  
  
  
    label_result = tk.Label(window, text=f"Predicted Label: {predicted_label}\nConfidence:  
{confidence:.2f}")  
  
    label_result.pack()
```

- **GUI Display:** Opens a new window (`Toplevel`) and displays the classified image along with the predicted label and confidence score.

upload_image()

```
def upload_image():  
    file_path = filedialog.askopenfilename()  
    if file_path:  
        classify_and_display(file_path)
```

File Upload: Opens a file dialog using `filedialog.askopenfilename()`, allowing the user to select an image file. If a file is selected, it invokes `classify_and_display()` to process and display the image.

. GUI Setup

```
root = tk.Tk()  
root.title("Image Classification with AlexNet")
```

```
btn_upload = tk.Button(root, text="Upload Image", command=upload_image)
btn_upload.pack(pady=20)
```

```
root.mainloop()
```

- **Tkinter GUI:** Creates the main application window (`root`), sets its title, and adds a button (`btn_upload`) that triggers the `upload_image()` function when clicked.
- **Main Event Loop:** Starts the main event loop (`root.mainloop()`), which listens for user interactions and keeps the GUI responsive.





Predicted Label: cheetah
Confidence: 0.99

This program combines deep learning (using AlexNet) for image classification with a graphical user interface built using Tkinter. Users can upload an image via a button, which triggers classification using the pretrained model, and displays the results in a new window. Error handling ensures robustness by managing potential exceptions during image processing.

TASK-11

AIM : LEARN LINEAR REGRESSION AND IMPLEMENT LINEAR REGRESSION TO PREDICT MOBLIE PRICE

OBJECTIVE

The objective of the provided Python program is to demonstrate a machine learning workflow for predicting mobile phone prices based on a dataset. Here are the specific objectives:

Data Loading and Verification:

- Load a dataset (`Mobile Price Prediction.csv`) containing information about mobile phones, including features and prices.
- Verify the presence of the 'Price (\$)' column in the dataset.

Data Preparation:

- Separate the dataset into features (X) and the target variable (y), where X includes all features except 'Price (\$)' and y represents the 'Price (\$)' column.

Data Splitting:

- Split the dataset into training and testing sets using a 70-30 split ratio (`train_test_split` function from `sklearn.model_selection`).
- Print the shapes of the training and testing sets to ensure proper division of data.

Model Training:

- Initialize a Linear Regression model (`LinearRegression` from `sklearn.linear_model`).
- Fit the model to the training data (`X_train`, `y_train`) to learn the relationship between features and prices.

Model Evaluation:

- Use the trained model to make predictions on the testing data (`X_test`).
- Evaluate the model's performance using the R-squared score (`r2_score` from `sklearn.metrics`), which measures how well the model predicts the variation in the target variable.

Prediction for New Data:

- Provide an example of how to predict the price of a new mobile phone with specified features (`new_mobile_data`).

Visualization:

- Create a scatter plot using `matplotlib` to visualize the actual vs predicted prices on the testing set (`y_test` vs `y_pred`).
- Include a reference line (`Perfect Prediction`) to visualize ideal predictions.

In summary, this program aims to showcase the complete pipeline of:

- Data loading and preparation,
- Model training and evaluation,
- Making predictions for new data,
- Visualizing results to assess model performance.

Ultimately, it demonstrates how machine learning can be used to predict mobile phone prices based on given features, facilitating decision-making in pricing strategies or market analysis.

DESCRIPTION

Import Necessary Libraries

```
import matplotlib

matplotlib.use('TkAgg') # Use TkAgg backend for interactive display

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score
```

Imports: Importing required libraries:

- `matplotlib`: For plotting graphs.
- `seaborn`: For enhancing visualizations.
- `pandas`: For data manipulation using DataFrames.
- `train_test_split` from `sklearn.model_selection`: For splitting data into training and testing sets.
- `LinearRegression` from `sklearn.linear_model`: For performing linear regression.
- `r2_score` from `sklearn.metrics`: For evaluating the model using R-squared score.

Load the Dataset

```
data = pd.read_csv("C:\\Users\\HP\\Pictures\\Mobile Price Prediction.csv")
```

Load Dataset: Reads a CSV file named "Mobile Price Prediction.csv" into a pandas DataFrame (data).

Verify and Prepare Data

```
print("Columns in dataset:", data.columns)
```

```
if 'Price ($)' in data.columns:
```

```
    X = data.drop(columns=['Price ($)']) # Features
```

```
    y = data['Price ($)'] # Target variable
```

```
else:
```

```
    print("Column 'Price' not found in dataset.")
```

Data Verification: Prints the columns in the dataset to verify that 'Price (\$)' is present. If present, separates the dataset into features (X) and the target variable (y). If 'Price (\$)' is not found, it prints an error message.

Split Data into Training and Testing Sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
print("X_train shape:", X_train.shape)
```

```
print("X_test shape:", X_test.shape)
```

```
print("y_train shape:", y_train.shape)
```

```
print("y_test shape:", y_test.shape)
```

Data Splitting: Splits the data into training and testing sets using `train_test_split` from `sklearn.model_selection`. Here:

- `X_train, y_train`: Training features and target.
- `X_test, y_test`: Testing features and target.
- `test_size=0.3`: 30% of the data is reserved for testing.
- `random_state=42`: Sets a seed for reproducibility.

Initialize and Train Linear Regression Model


```
linreg = LinearRegression()
```

```
linreg.fit(X_train, y_train)
```

Model Initialization and Training: Initializes a Linear Regression model (`linreg`) and fits it to the training data (`X_train, y_train`).

Make Predictions and Evaluate Model

```
y_pred = linreg.predict(X_test)
```

```
r2 = r2_score(y_test, y_pred)
```

```
print("R-squared score:", r2)
```

- **Prediction and Evaluation:** Uses the trained model to make predictions (`y_pred`) on the testing data (`X_test`) and evaluates the model's performance using the R-squared score (`r2_score`).

Example Prediction for a New Mobile Phone

```
new_mobile_data = {  
    'Screen Size (inches)': 6.5,  
    'RAM (GB)': 4,  
    'Storage (GB)': 128,  
    'Battery Capacity (mAh)': 4000,  
    'Camera Quality (MP)': 48  
}
```

```
new_mobile_df = pd.DataFrame(new_mobile_data, index=[0])
```

```
price_pred = linreg.predict(new_mobile_df)  
print("Predicted Price of the New Mobile Phone:", price_pred)
```

New Mobile Prediction: Creates a DataFrame (`new_mobile_df`) containing data for a hypothetical new mobile phone and predicts its price using the trained model (`linreg`).

Plotting - Visualizing the Results

```
plt.figure(figsize=(10, 6))
```

```

plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='red',
label='Perfect Prediction')
plt.title('Actual vs Predicted Prices')
plt.xlabel('Actual Price ($)')
plt.ylabel('Predicted Price ($)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

```

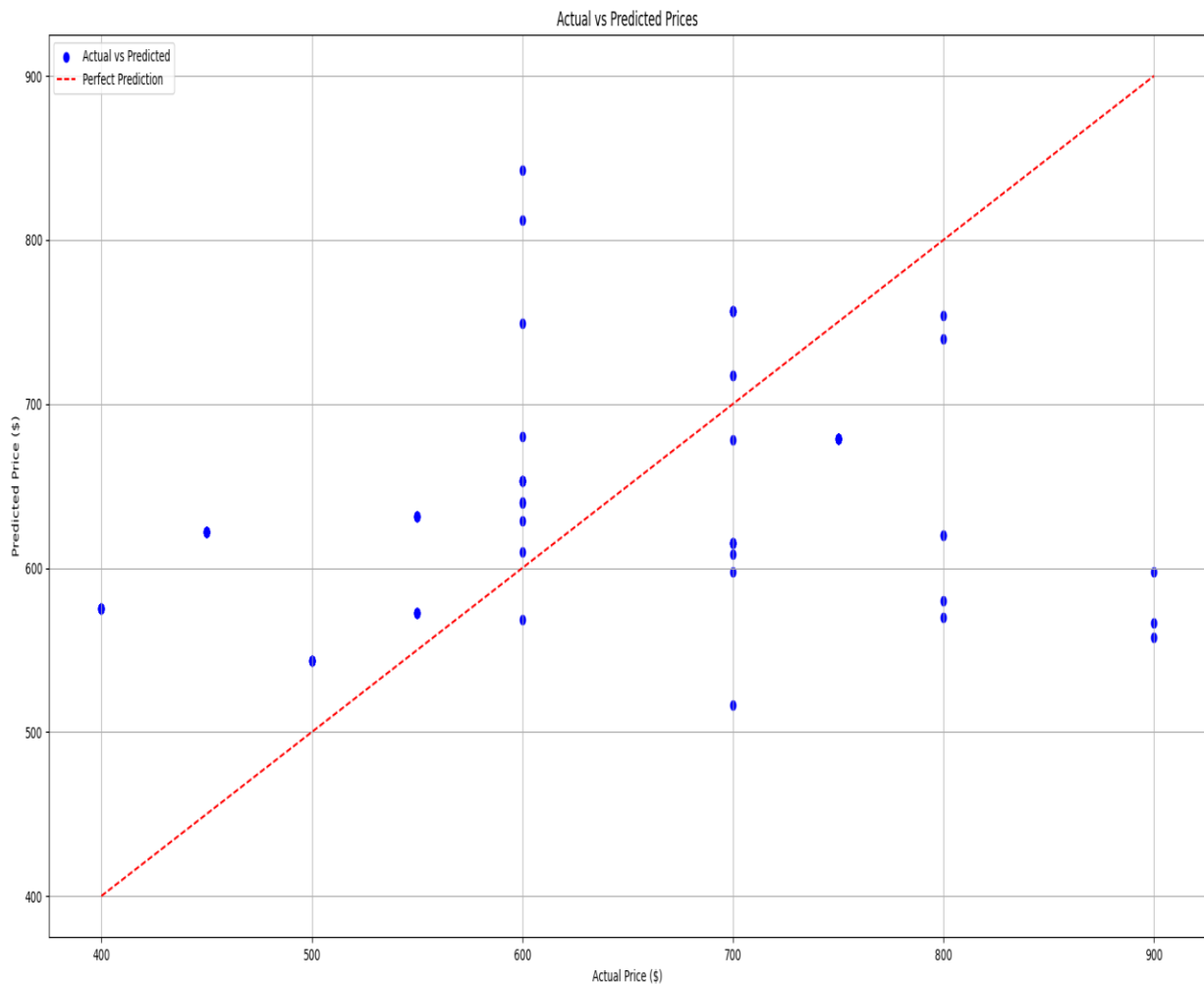
Visualization: Generates a scatter plot to visualize the actual vs predicted prices (`y_test` vs `y_pred`). Also includes a red dashed line (**Perfect Prediction**) indicating a perfect prediction scenario.

```

C:\Users\HP\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:\Users\HP\PycharmProjects\pythonProject1\CAI_006_CLASS\linearregression_final.py
Columns in dataset: Index(['Screen Size (inches)', 'RAM (GB)', 'Storage (GB)',
    'Battery Capacity (mAh)', 'Camera Quality (MP)', 'Price ($)'],
    dtype='object')
X_train shape: (204, 5)
X_test shape: (88, 5)
y_train shape: (204,)
y_test shape: (88,)
R-squared score: 0.07068251520957014
Predicted Price of the New Mobile Phone: [552.80889158]

Process finished with exit code 0

```



This program demonstrates the process of:

- Loading and preparing data for regression analysis.
- Training a linear regression model to predict mobile phone prices based on features.
- Evaluating the model's performance using R-squared score.
- Making predictions for new data and visualizing results using matplotlib.

It serves as a practical example of applying machine learning techniques to real-world datasets for predictive modeling and visualization.

CONCLUSION

My internship provided me with a comprehensive toolkit in image processing, machine learning, and object detection, facilitated by a range of essential tools and technologies. I adeptly utilized OpenCV and PIL/Pillow for foundational image processing tasks, while TensorFlow and Keras empowered me to construct and train Convolutional Neural Networks (CNNs) for image classification. I applied advanced techniques such as thresholding with OpenCV and character extraction using Tesseract OCR, demonstrating proficiency in segmentation methods like SIFT and histogram matching for image analysis. Object detection tasks were efficiently handled using pre-trained models like AlexNet integrated with torchvision and PyTorch. Additionally, I developed predictive models using scikit-learn for linear regression to forecast mobile prices based on diverse features. My proficiency in data manipulation and visualization using NumPy, Pandas, Matplotlib, and Seaborn, coupled with GUI development using Tkinter, underscored my ability to translate theoretical knowledge into practical solutions. This internship significantly enhanced my expertise in leveraging cutting-edge tools and technologies to solve complex challenges in computer vision and machine learning domains.

REFERENCES

BASIC IMAGE PROCESSING -

<https://www.geeksforgeeks.org/image-processing-in-python/>

CAT AND DOG IMAGE CLASSIFICATION-

<https://www.kaggle.com/akrsnv>

<https://www.analyticsvidhya.com/blog/2021/06/beginner-friendly-project-cat-and-dog-classification-using-cnn/>

THRESHOLD-

<https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/>

SHIFT- <https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/>

WATER SHED SEGMENTATION TECHNIQUE-

<https://www.geeksforgeeks.org/image-segmentation-with-watershed-algorithm-opencv-python/>

REGION_BASE_SEGMENTATION-

<https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/>

CANNY_EDGE_DETECTION<https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/>

LINEAR REGRESSION FOR MOBILE PRICE PREDICTION-

<https://www.kaggle.com/datasets/vishwajeetvijaykale/mobile-price-prediction-very-beginner-friendly>

<https://www.analyticsvidhya.com/blog/2022/02/learn-mobile-price-prediction-through-four-classification-algorithms/>

OBJECT DETECTION USING ALREADY EXISTING MODEL-

<https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json>

<https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/>