Pro:1

1. Import OpenCV library.
2. Load image from file using cv2.imread().
3. Convert the image to grayscale using cv2.cvtColor().
4. Display the grayscale image using cv2.imshow().
5. Wait for keyboard event using cv2.waitKey().
6. Save the grayscale image to file using cv2.imwrite().

Pro:2

1. import cv2 and numpy libraries
2. set the file path to the image file
3. read the image using cv2.imread() function and store it in img variable
4. access a pixel value from the grayscale image by specifying the row and column index along with the color channel (0 for grayscale)
5. print the pixel value
6. modify the pixel value at the specified index to a new value (e.g. 255 for maximum intensity)
7. access the new pixel value and print it
8. save the modified image using cv2.imwrite() function

pro:3

1. Load an image from the specified file path using cv2.imread() function.
2. Resize the image using cv2.resize() function with the new width and height as parameters.
3. Create a rotation matrix using cv2.getRotationMatrix2D() function with the center point of the image, rotation angle and scale as parameters.
4. Rotate the image using cv2.warpAffine() function with the original image, rotation matrix and size of the output image as parameters.
5. Display the resized and rotated images using cv2.imshow() function.
6. Wait for the user to press any key to close the windows using cv2.waitKey() function with 0 as a parameter.

Pro:4

1. Import the OpenCV library.
2. Load two images using cv2.imread() function and store them in two separate variables.
3. Perform image addition using cv2.addWeighted() function by providing the two images, the weight for each image, and a scalar value.
4. Display the resulting image using cv2.imshow() function.
5. Wait for the user to close the window using cv2.waitKey(0) function.

Pro:5

1. Read in an image using OpenCV imread function
2. Apply Mean Filter on the image using OpenCV blur function
3. Display the original image and the resulting image using matplotlib subplot function
4. Read in another image using OpenCV imread function
5. Apply Gaussian Filter on the image using OpenCV GaussianBlur function
6. Display both original and blurred images using OpenCV imshow and numpy hstack functions
7. Wait for a keyboard event to exit the program using OpenCV waitKey function
8. Close all windows using OpenCV destroyAllWindows function

Pro:6

1. Import the required libraries: cv2 and numpy.
2. Read the image using cv2.imread() function.
3. Convert the image to grayscale using cv2.cvtColor() function.
4. Set a threshold value using cv2.threshold() function.
5. Display the thresholded image using cv2.imshow() function.
6. Wait for a key press using cv2.waitKey() function.
7. Close all the windows using cv2.destroyAllWindows() function.

Pro:8

1. Read the image using cv2.imread() function and store it in a variable.
2. Use the cv2.cvtColor() function to convert the image from BGR to RGB color space and store it in a variable.
3. Use the matplotlib.pyplot.imshow() function to display the original image without the axis.
4. Use the cv2.calcHist() function to calculate the histogram of the image for each color channel (blue, green, red) and store them in separate variables.
5. Use the matplotlib.pyplot.hist() function to display the histogram of each color channel separately with different colors.

Pro:9

1. Import **numpy** and **cv2**.
2. Load the image using **cv2.imread()**.
3. Reshape and convert the image data type to **float32**.
4. Define stopping criteria for K-means algorithm.

5. Set the number of clusters as 8.
6. Apply K-means clustering using **cv2.kmeans()**.
7. Convert the center values to **uint8**.
8. Apply the clustering to the image and reshape the output image.
9. Display the output image using **cv2.imshow()**.
10. Wait for the user to close the window using **cv2.waitKey()**.
11. Destroy all windows using **cv2.destroyAllWindows()**.

Pro:7

1. Import the necessary libraries: cv2 and numpy
2. Load the image in grayscale using cv2.imread
3. Apply the Sobel operator for edge detection in the x direction using cv2.Sobel
4. Apply the Sobel operator for edge detection in the y direction using cv2.Sobel
5. Apply the Scharr operator for edge detection in the x direction using cv2.Scharr
6. Apply the Scharr operator for edge detection in the y direction using cv2.Scharr
7. Display the original image and the edge-detected images using cv2.imshow
8. Wait for a key press using cv2.waitKey
9. Close all windows using cv2.destroyAllWindows.

Pro:10

1. Load the digits dataset using load_digits() from sklearn.datasets.
2. Split the dataset into training and testing data using train_test_split() from sklearn.model_selection.
3. Initialize a KNN classifier using KNeighborsClassifier() from sklearn.neighbors.
4. Train the classifier using fit() method with the training data.
5. Predict the labels of testing data using predict() method.
6. Calculate the accuracy of the model using accuracy_score() from sklearn.metrics.
7. Create a confusion matrix using confusion_matrix() from sklearn.metrics.
8. Visualize the confusion matrix using heatmap() from seaborn with annot=True, fmt='.3f', linewidths=.5, square=True, and cmap='Blues_r'.
9. Add x-label, y-label, and title to the heatmap using xlabel(), ylabel(), and title() methods of matplotlib.pyplot.