PATTERN RECOGNITION & COMPUTER VISION

ML – 411P

Faculty Name:
Ms. Prachi Dahiya

Made By: Vanshik Sanwaria
Roll No.: 01914812721
Semester: 7th
Batch: CST (AIML)

Maharaja Agrasen Institute of Technology, PSP area, Sector-22, Rohini, New Delhi -110085

Department of Computer Science and Engineering

Rubrics for Lab Assessment

Rubrics		0	1	2	3
		Missing	**Inadequate**	**Needs Improvement**	**Adequate**
R1	Is able to identify the problem to be solved and define the objectives of the experiment.	No mention is made of the problem to be solved.	An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable.	The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors.	The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors.
R2	Is able to design a reliable experiment that solves the problem.	The experiment does not solve the problem.	The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution.	The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable solution.	The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution.
R3	Is able to communicate the details of an experimental procedure clearly and completely.	Diagrams are missing and/or experimental procedure is missing or extremely vague.	Diagrams are present but unclear and/or experimental procedure is present but important details are missing.	Diagrams and/or experimental procedure are present but with minor omissions or vague details.	Diagrams and/or experimental procedure are clear and complete.
R4	Is able to record and represent data in a meaningful way.	Data are either absent or incomprehensible.	Some important data are absent or incomprehensible.	All important data are present, but recorded in a way that requires some effort to comprehend.	All important data are present, organized and recorded clearly.
R5	Is able to make a judgment about the results of the experiment.	No discussion is presented about the results of the experiment .	A judgment is made about the results, but it is not reasonable or coherent.	An acceptable judgment is made about the result, but the reasoning is flawed or incomplete.	An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered.

INDEX

Name: Vanshik Sanwaria
Branch: Computer Science and Technology
Roll No: 01914812721
Group: 7 CST - 1

S No.	Experiment	Rubrics					Date of performing	Signature
		R1	R2	R3	R4	R5		
1.	Write a MATLAB/Python function that computes the value of the Gaussian distribution N(m,s) at given vector X and plot the effect of varying mean and variance to the normal distribution.							
2.	Implementation of Gradient descent.							
3.	Implementation of Linear Regression using Gradient descent.							
4.	Comparison of classification accuracy of SVM and CNN for the dataset.							
5.	Implementation basic Image Handling and processing operations on the image							
6.	Implementation of Geometric Transformation.							
7.	Implementation of Perspective Transformation.							
8.	Implementation of Camera Calibration							
9.	Compute Fundamental Matrix							

Experiment-1

Aim: Write a Python function to compute the value of the Gaussian distribution N(m,s)N(m, s)N(m,s) and plot varying mean and variance.

Theory: The Gaussian distribution (also known as the normal distribution) is a continuous probability distribution characterized by two parameters: mean μ and variance σ^2. The formula for the Gaussian distribution is:

$$N(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

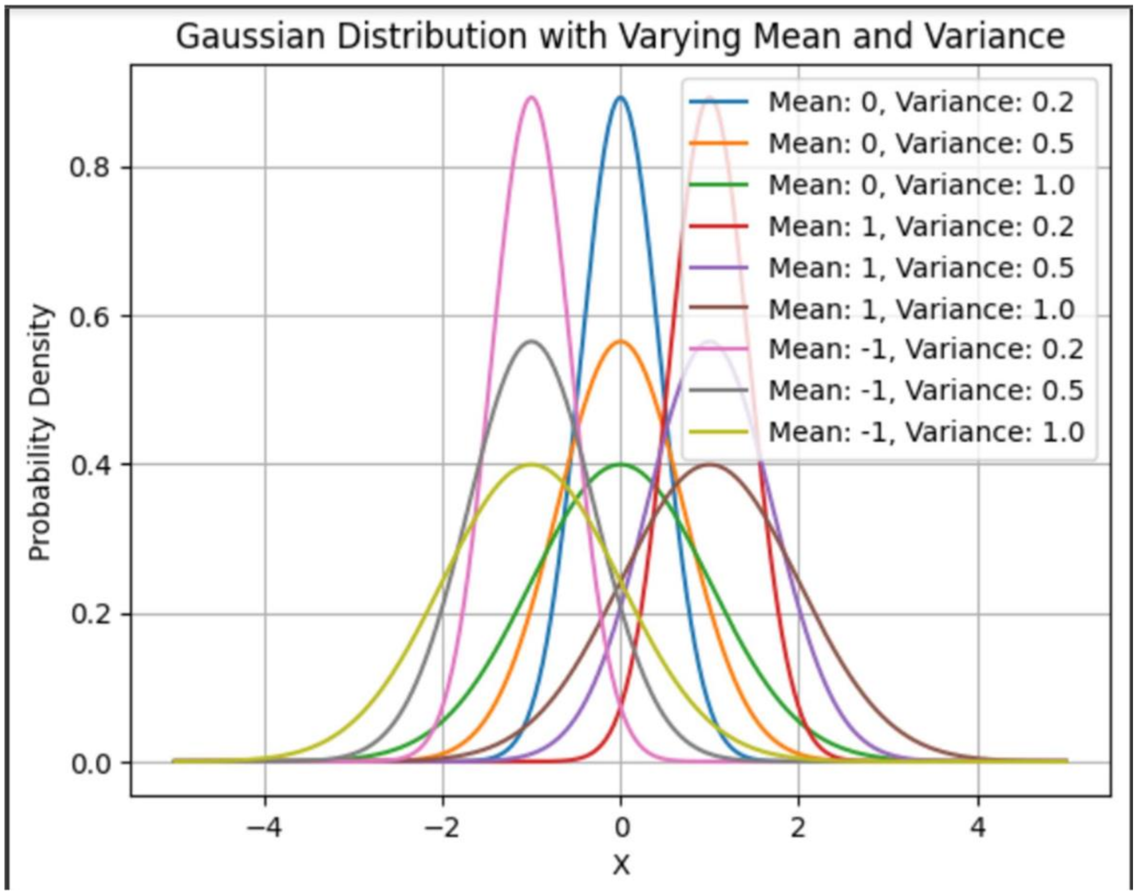
The mean μ determines the central location of the curve, while the variance σ2 controls its spread. In this practical, we compute the value of the Gaussian distribution for different means and variances, plotting the results to visualize how these parameters affect the distribution. The distribution is symmetric around the mean, with most values concentrated around it, and the curve's width increases with higher variance.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt  #

Gaussian distribution function

def gaussian_distribution(x, mean, variance):

   return (1 / (np.sqrt(2 * np.pi * variance))) * np.exp(-(x - mean)**2 / (2 * variance))

# Plotting function

def plot_gaussian(mean_values, variance_values, x_range):  x

   = np.linspace(*x_range, 1000)

   for mean in mean_values:

      for variance in variance_values:

         y = gaussian_distribution(x, mean, variance)

         plt.plot(x, y, label=f'Mean: {mean}, Variance: {variance}')

   plt.title('Gaussian Distribution with Varying Mean and Variance')

   plt.xlabel('X')

   plt.ylabel('Probability Density')

   plt.legend()

   plt.grid(True)
```

```
# Parameters

mean_values = [0, 1, -1]

variance_values = [0.2, 0.5, 1.0]

x_range = (-5, 5)


# Plot Gaussian distributions

plot_gaussian(mean_values, variance_values, x_range)
```

Output:



Viva Question

1. What is a Gaussian distribution?

- The Gaussian distribution, also known as the normal distribution, is a continuous probability distribution characterized by its bell-shaped curve. It is defined by two parameters: the mean (m) and the variance (s^2).

2. How does the mean affect the shape of the Gaussian distribution?

- The mean determines the location of the center of the distribution. A higher or lower mean shifts the bell curve left or right along the x-axis.

3. What role does the variance play in the Gaussian distribution?

- The variance determines the spread of the distribution. A higher variance results in a wider curve, while a smaller variance makes the curve narrower.

4. Why is the Gaussian distribution important in statistics and machine learning?

- The Gaussian distribution is widely used because of the central limit theorem, which states that the sum of many independent random variables tends to follow a normal distribution, regardless of their original distributions. It also simplifies many statistical methods and algorithms.

5. How can you compute the Gaussian distribution using Python?

- Using the formula: N(x,m,s)=12πsexp (−(x−m)22s)N(x, m, s) = \frac{1}{\sqrt{2\pi s}} \exp\left(\frac{-(x - m)^2}{2s}\right)N(x,m,s)=2πs1 exp(2s−(x−m)2) This can be implemented with NumPy to handle vectorized operations efficiently, as shown in the function above.

Experiment-2

Aim: Implement Gradient
Descent Function.

Theory: Gradient descent is an optimization algorithm used to minimize the cost function by iteratively adjusting parameters in the direction of the steepest descent (negative gradient). It’s widely used in machine learning for finding the best parameters in regression and classification tasks. The algorithm starts with random parameter values, computes the gradient (partial derivatives) of the cost function, and updates the parameters using the formula:

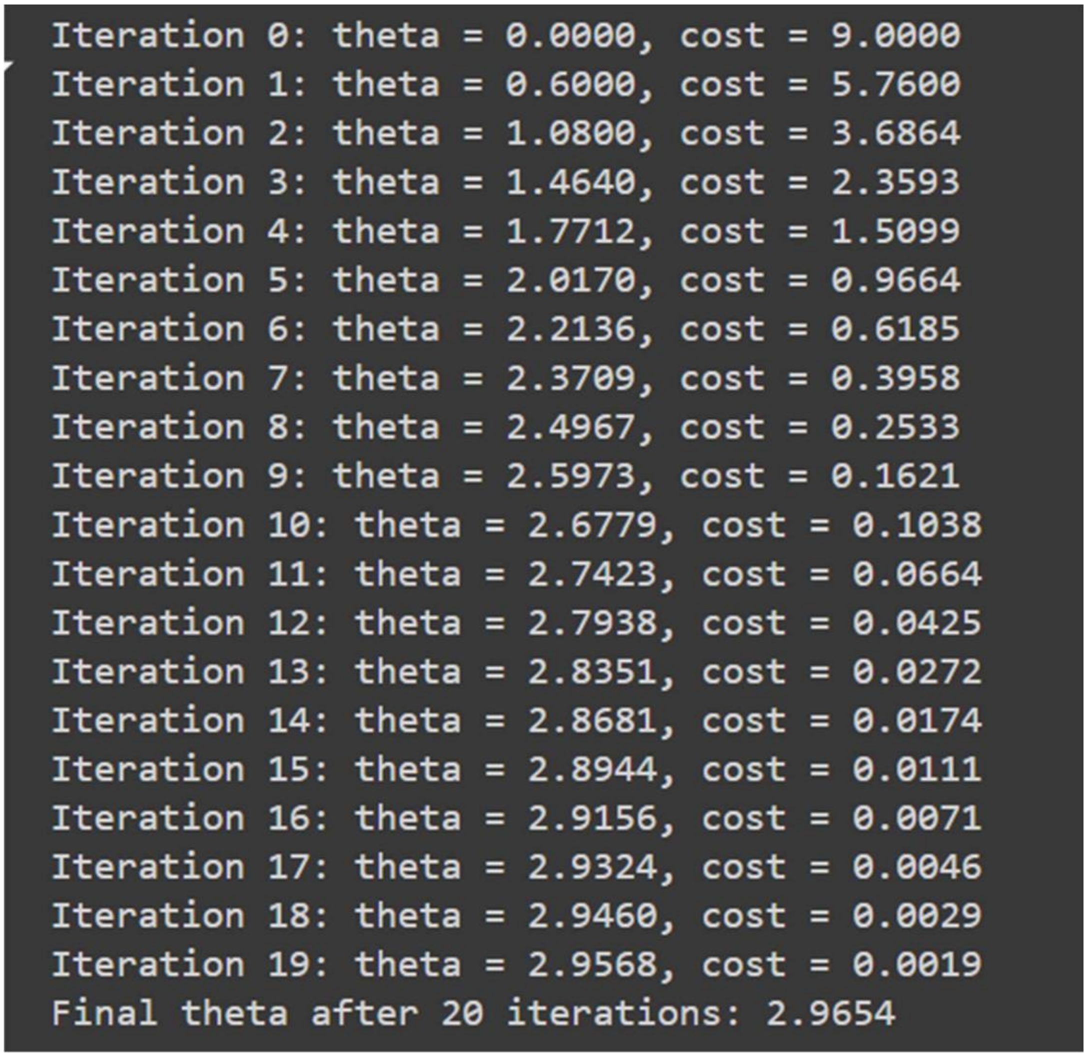
$$\theta = \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$$

Where α is the learning rate, and J(θ) is the cost function.

Source Code:

```
import numpy as np

# Example cost function: J(theta) = (theta - 3)^2
def cost_function(theta):
    return (theta - 3) ** 2

# Derivative of the cost function: dJ/dtheta = 2 * (theta - 3)
def gradient(theta):
    return 2 * (theta - 3)

# Gradient Descent Algorithm
def gradient_descent(initial_theta, learning_rate, iterations):
    theta = initial_theta
    history = []
    for _ in range(iterations):
        cost = cost_function(theta)
        history.append((theta, cost))
        theta = theta - learning_rate * gradient(theta)
    return theta, history

# Parameters
initial_theta = 0.0
learning_rate = 0.1
iterations = 20

# Perform gradient descent
final_theta, history = gradient_descent(initial_theta, learning_rate, iterations)

# Display results
for step, (theta, cost) in enumerate(history):
    print(f"Iteration {step}: theta = {theta:.4f}, cost = {cost:.4f}")

print(f"Final theta after {iterations} iterations: {final_theta:.4f}")
```

Output:



```
Iteration 0: theta = 0.0000, cost = 9.0000
Iteration 1: theta = 0.6000, cost = 5.7600
Iteration 2: theta = 1.0800, cost = 3.6864
Iteration 3: theta = 1.4640, cost = 2.3593
Iteration 4: theta = 1.7712, cost = 1.5099
Iteration 5: theta = 2.0170, cost = 0.9664
Iteration 6: theta = 2.2136, cost = 0.6185
Iteration 7: theta = 2.3709, cost = 0.3958
Iteration 8: theta = 2.4967, cost = 0.2533
Iteration 9: theta = 2.5973, cost = 0.1621
Iteration 10: theta = 2.6779, cost = 0.1038
Iteration 11: theta = 2.7423, cost = 0.0664
Iteration 12: theta = 2.7938, cost = 0.0425
Iteration 13: theta = 2.8351, cost = 0.0272
Iteration 14: theta = 2.8681, cost = 0.0174
Iteration 15: theta = 2.8944, cost = 0.0111
Iteration 16: theta = 2.9156, cost = 0.0071
Iteration 17: theta = 2.9324, cost = 0.0046
Iteration 18: theta = 2.9460, cost = 0.0029
Iteration 19: theta = 2.9568, cost = 0.0019
Final theta after 20 iterations: 2.9654
```

Viva Questions

1. What is gradient descent, and why is it used?

o Gradient descent is an optimization algorithm used to minimize a cost function by iteratively adjusting model parameters in the opposite direction of the gradient of the cost function with respect to the parameters.

2. What does the learning rate (α) represent in gradient descent?

o The learning rate controls how large the steps are in each iteration towards the minimum. A small learning rate results in slow convergence, while a large one may overshoot the minimum.

3. What are the conditions under which gradient descent might fail to converge?

o Gradient descent may fail to converge if the learning rate is too high, causing the algorithm to oscillate, or if the cost function is not well- behaved (e.g., has many local minima).

4. How is the cost function related to gradient descent?

o The cost function quantifies the error between the predicted and actual values. Gradient descent minimizes the cost function by adjusting the parameters in the direction where the cost decreases.

5. How do you decide the number of iterations in gradient descent?

o The number of iterations can be decided by monitoring the change in the cost function. When the cost change between iterations is negligible, convergence can be assumed, and the iterations can be stopped.

Experiment-3

Aim: Implement Linear Regression using Gradient Descent

Theory: Linear regression is a supervised learning algorithm used to predict a continuous target variable based on one or more input features. The relationship between the input (X) and output (Y) is modelled by a straight line:

$$Y = \theta_0 + \theta_1 X$$

Here, θ0 is the intercept and θ1 is the slope. Gradient descent is used to minimize the cost function, typically the Mean Squared Error (MSE), which measures how far the predicted values are from the actual values. The cost function for linear regression is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - Y^{(i)})^2$$

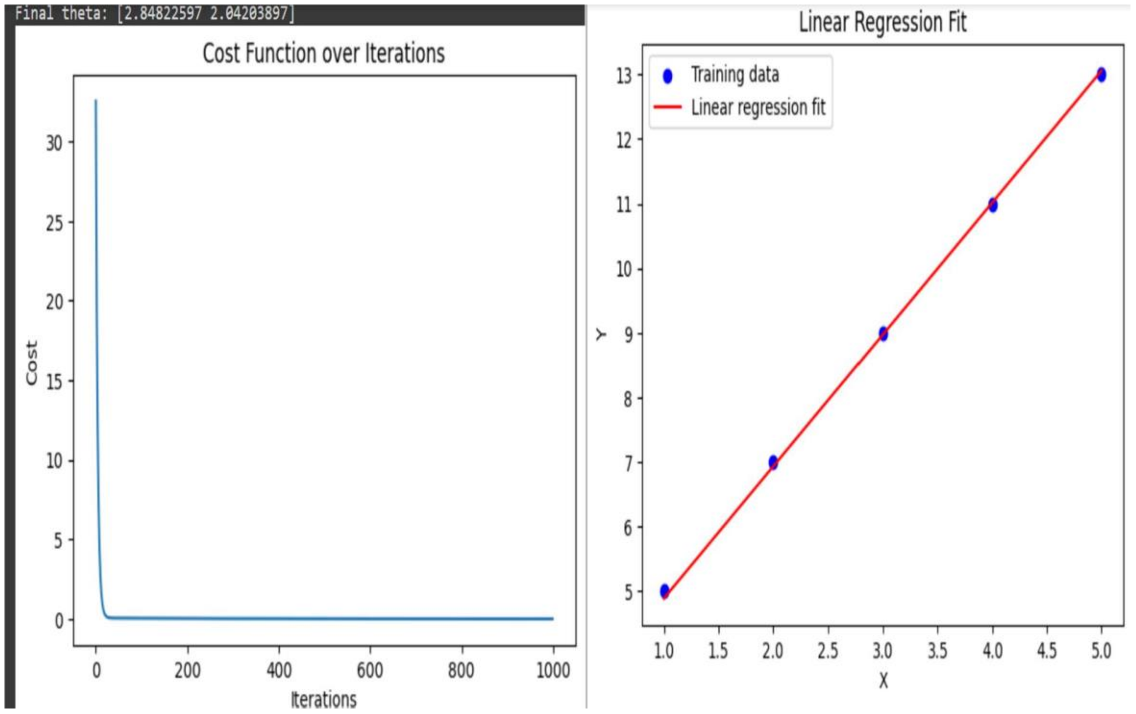
where hθ(X) is the predicted value, and m is the number of training examples.

The gradient descent updates the parameters iteratively until it finds the optimal values that minimize the cost function.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt

# Hypothesis (Prediction function)
def predict(X, theta):
    return np.dot(X, theta)

# Cost function (Mean Squared Error)
def compute_cost(X, Y, theta):
    m = len(Y)
    return (1/(2*m)) * np.sum((predict(X, theta) - Y)**2)

# Gradient Descent Algorithm
def gradient_descent(X, Y, theta, learning_rate, iterations):
    m = len(Y)
    cost_history = []

    for _ in range(iterations):
        theta = theta - (learning_rate/m) * np.dot(X.T, predict(X, theta) - Y)
        cost_history.append(compute_cost(X, Y, theta))
    return theta, cost_history

# Data (example)
# Add a bias (intercept) term to X (X_0 = 1)
X_b = np.c_[np.ones((len(X), 1)), X]  # Add a column of 1s for theta_0

# Initialize theta (parameters)
theta = np.random.randn(2)

# Parameters
learning_rate = 0.01
iterations = 1000

# Run gradient descent
final_theta, cost_history = gradient_descent(X_b, Y, theta, learning_rate, iterations)

# Display final parameters
print(f"Final theta: {final_theta}")

# Plot the cost function history
plt.plot(cost_history)
plt.title('Cost Function over Iterations')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()

# Plot the fitted line
plt.scatter(X, Y, color='blue', label='Training data')
plt.plot(X, predict(X_b, final_theta), color='red', label='Linear regression fit')
plt.title('Linear Regression Fit')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Output:



```
Final theta: [2.84822597 2.04203897]
```

Viva Questions

1. What is linear regression, and how is it different from other regression models?

o Linear regression models the relationship between variables as a straight line, assuming the relationship is linear, while other regression models (like polynomial or logistic regression) can model more complex relationships.

2. What is the role of the cost function in linear regression?

o The cost function (mean squared error in this case) quantifies the error between predicted and actual values. Gradient descent minimizes this cost function to optimize the model parameters.

3. Why do we need to add a bias term in linear regression?

o The bias term allows the model to fit data that does not pass through the origin. Without it, the line is forced to pass through the origin, which might not be the best fit.

4. What happens if the learning rate is too high or too low in gradient descent?

o A high learning rate may cause the algorithm to overshoot the minimum and fail to converge, while a low learning rate results in slow convergence and increases the computation time.

5. What is the significance of the number of iterations in gradient descent?

o The number of iterations determines how many times the parameters will be updated. If the number is too low, the algorithm might stop before reaching the optimal values.

Experiment-4

Aim: Comparison of Classification Accuracy: SVM vs CNN

Theory: Support Vector Machines (SVMs) and Convolutional Neural Networks (CNNs) are both popular algorithms for classification tasks. SVM is a discriminative classifier that finds a hyperplane separating classes in high-dimensional space. It works well on smaller datasets and structured data. On the other hand, CNNs are deep learning models that are particularly suited for image classification due to their ability to capture spatial hierarchies through convolutional layers.

For this experiment, we'll use a common dataset (e.g., MNIST) and compare the accuracy of SVM and CNN classifiers.

Source Code:

- SVM

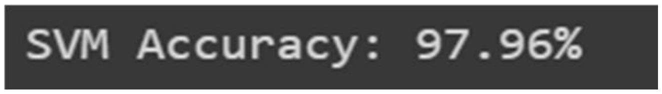
```
 from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load dataset (e.g., digits dataset)
digits = datasets.load_digits()
X = digits.images.reshape((len(digits.images), -1))  # Flatten the images
Y = digits.target

# Split dataset 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)

# Train SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, Y_train)

# Test SVM model
Y_pred_svm = svm_model.predict(X_test)

# SVM Accuracy
svm_accuracy = accuracy_score(Y_test, Y_pred_svm)
print(f"SVM Accuracy: {svm_accuracy * 100:.2f}%")
```

- CNN

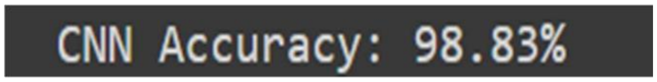
```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split

# Load dataset (e.g., MNIST dataset)
(X_train, Y_train), (X_test, Y_test) = tf.keras.datasets.mnist.load_data()

# Normalize the data
X_train = X_train / 255.0
X_test = X_test / 255.0

# Build CNN model
```

```
cnn_model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Reshape the data for the CNN
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# Train CNN model
cnn_model.fit(X_train, Y_train, epochs=5)

# Test CNN model
cnn_accuracy = cnn_model.evaluate(X_test, Y_test, verbose=0)
print(f"CNN Accuracy: {cnn_accuracy[1] * 100:.2f}%")
```

Output:



```
SVM Accuracy: 97.96%
```



```
CNN Accuracy: 98.83%
```

Viva Questions

1. What is the key difference between SVM and CNN in terms of their approach to classification tasks?
o This question assesses your understanding of the foundational difference between SVM's mathematical boundary-based approach and CNN's feature-learning through convolutions.
2. How does the concept of kernel trick in SVMs help in classifying non- linearly separable data?
o This explores the use of kernel functions in SVM to handle complex data and improve accuracy for non-linear problems.
3. Why do CNNs generally outperform SVMs on image classification tasks?
o This focuses on the ability of CNNs to automatically learn hierarchical features from images, which SVM lacks unless combined with pre- processing and manual feature extraction.
4. What are the computational requirements of CNN compared to SVM, and how does dataset size impact both models?
o This probes your understanding of the trade-offs in computational complexity, training time, and data requirements for both algorithms.
5. Can SVMs be combined with CNNs in any practical scenarios, and if so, how?
o This question checks if you know about hybrid models, where CNNs are used for feature extraction and SVMs for classification, a technique sometimes used in research or specific use cases.

Experiment-5

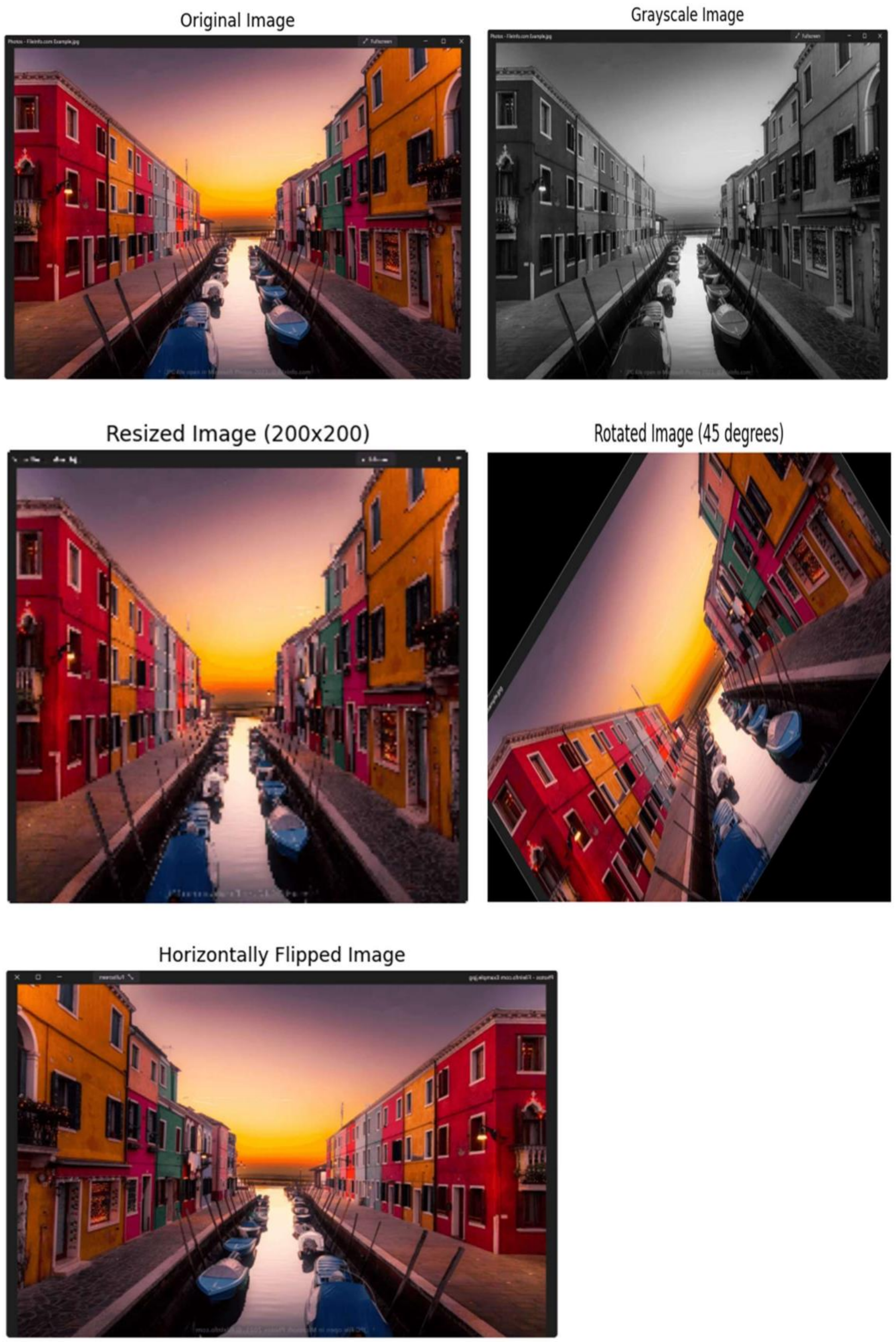
Aim: Implement Basic Image Handling and Processing operations on the Image

Theory: Image handling and processing involve tasks such as loading, displaying, and manipulating images, which are crucial in computer vision. These tasks can be performed using libraries like OpenCV or PIL (Pillow). Common operations include resizing, rotating, flipping, converting between color spaces (e.g., RGB to grayscale), and drawing shapes on images. These basic techniques form the foundation for more advanced image processing tasks, such as filtering, edge detection, and segmentation, which are often used in object detection and recognition systems.

Source Code:

```
 import cv2
from matplotlib import pyplot as plt

# Load an image
image = cv2.imread(' /content/a.jpg')

# Convert the image from BGR to RGB (since OpenCV loads in BGR)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the original image
plt.imshow(image_rgb)
plt.title('Original Image')
plt.axis('off')  # Turn off axis numbers
plt.show()

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_image, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')
plt.show()

# Resize the image
resized_image = cv2.resize(image_rgb, (200, 200))
plt.imshow(resized_image)
plt.title('Resized Image (200x200)')
plt.axis('off')
plt.show()

# Rotate the image
(h, w) = image.shape[:2]
center = (w // 2, h // 2)
matrix = cv2.getRotationMatrix2D(center, 45, 1.0)  # Rotate by 45 degrees
rotated_image = cv2.warpAffine(image_rgb, matrix, (w, h))
plt.imshow(rotated_image)
plt.title('Rotated Image (45 degrees)')
plt.axis('off')
plt.show()

# Flip the image (horizontal flip)
flipped_image = cv2.flip(image_rgb, 1)
plt.imshow(flipped_image)
plt.title('Horizontally Flipped Image')
plt.axis('off')
plt.show()
```

Output:


Original Image
Grayscale Image
Resized Image (200x200)
Rotated Image (45 degrees)
Horizontally Flipped Image

Viva Questions

1. What are the most common image formats you encounter, and how does each format differ in terms of usage and compression?

- This tests your knowledge of formats like JPEG, PNG, BMP, and their respective uses in terms of quality, compression, and file size.

2. What are the basic operations involved in image preprocessing, and why are they important?

- This explores your understanding of operations such as resizing, cropping, grayscale conversion, and their role in preparing an image for further processing or analysis.

3. Can you explain the concept of image filtering and name a few commonly used filters?

- The goal is to check your understanding of image filters like Gaussian blur, sharpening, and edge detection, and how they are applied to enhance or extract features from an image.

4. How does converting an image to grayscale help in simplifying image processing tasks?

- This question focuses on the advantage of reducing color complexity when applying algorithms that don't require color information, such as edge detection or object recognition.

5. What are some common challenges in image processing, and how do you address them?

- This assesses your practical experience with challenges like noise, lighting variations, or low-resolution images, and how you use techniques like denoising, histogram equalization, or super-resolution to handle them.

Experiment-6

Aim: Implementation of Geometric Transformation

Theory: Geometric transformations involve altering the spatial arrangement of pixels in an image through operations like scaling, translation, rotation, and affine transformations. These transformations are essential in image manipulation tasks such as resizing or aligning images. Each transformation can be represented by a matrix, which when applied to the image's coordinates, alters its structure.

- **Scaling** changes the size of the image.
- **Translation** shifts the image along the x or y axis.
- **Rotation** rotates the image by a specified angle.
- **Affine transformations** preserve parallelism but not distances and angles, useful for shearing or skewing images.

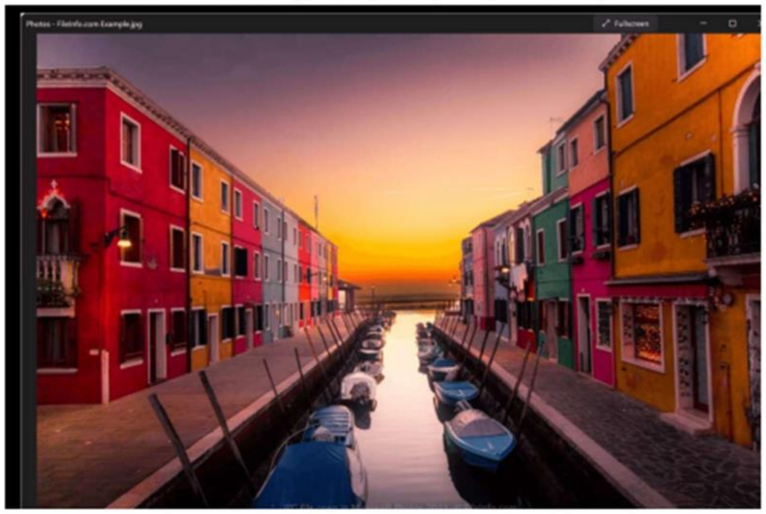
Source Code:

```
 # Scaling the image
scaled_image = cv2.resize(image_rgb, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)
plt.imshow(scaled_image)
plt.title('Scaled Image (150%)')
plt.axis('off')
plt.show()

# Translating the image (shift by 50 pixels right and 30 pixels down)
translation_matrix = np.float32([[1, 0, 50], [0, 1, 30]])
translated_image = cv2.warpAffine(image_rgb, translation_matrix, (w, h))
plt.imshow(translated_image)
plt.title('Translated Image (Right by 50, Down by 30)')
plt.axis('off')
plt.show()

# Rotating the image (already covered earlier)

# Affine Transformation (Shearing effect)
points_before = np.float32([[50, 50], [200, 50], [50, 200]])
points_after = np.float32([[10, 100], [200, 50], [100, 250]])
affine_matrix = cv2.getAffineTransform(points_before, points_after)
affine_image = cv2.warpAffine(image_rgb, affine_matrix, (w, h))
plt.imshow(affine_image)
plt.title('Affine Transformed Image (Shearing)')
plt.axis('off')
plt.show()
```

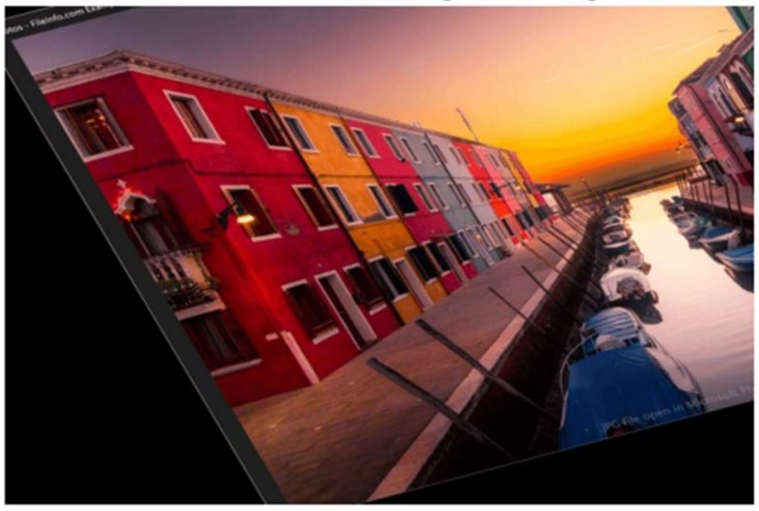
Output:

Scaled Image (150%)

Translated Image (Right by 50, Down by 30)



Affine Transformed Image (Shearing)



Viva Questions

1. What is geometric transformation in image processing, and why is it used?

- This tests your understanding of geometric transformation concepts and the purpose of altering the position, orientation, or shape of an image.

2. Can you explain the differences between affine and non-affine transformations?

- This question is to assess your knowledge of specific transformations like translation, rotation, scaling (affine), and more complex ones like warping (non-affine).

3. What are the mathematical principles behind translation, rotation, and scaling transformations?

- This focuses on your understanding of transformation matrices and how each geometric transformation is applied mathematically to the coordinates of an image.

4. How does interpolation affect image quality during geometric transformations, and which interpolation methods are commonly used?

- This explores your understanding of the importance of interpolation methods like nearest-neighbor, bilinear, and bicubic during transformations and how they influence the resulting image quality.

5. What challenges can arise when applying geometric transformations, and how would you handle issues like image distortion or loss of information?

- This assesses your practical experience with handling problems such as pixelation, aspect ratio distortion, or boundary effects that can occur during geometric transformations, and how to mitigate them using techniques like padding

Experiment-7

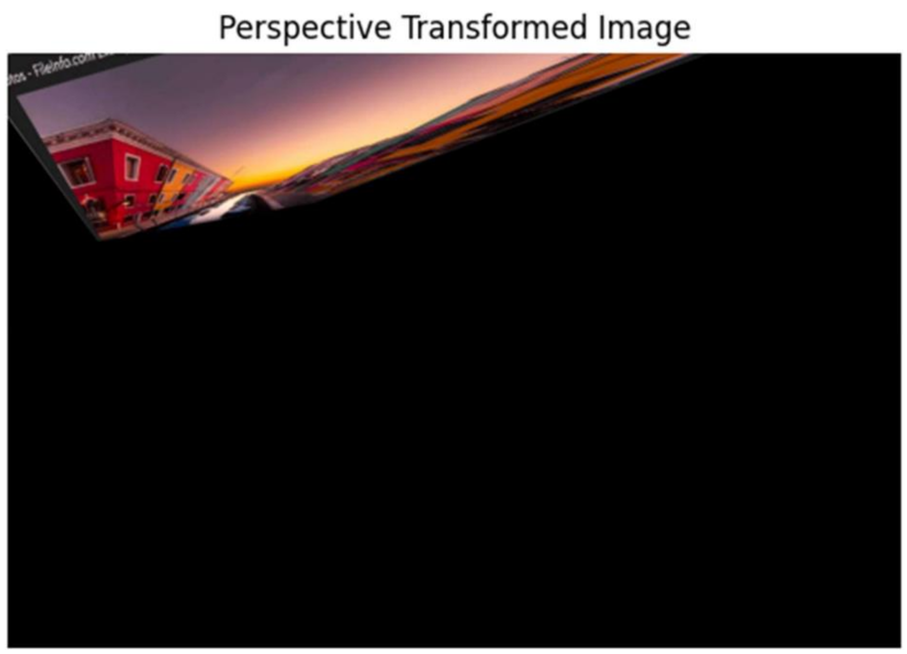
Aim: Implementation of Perspective Transformation

Theory: Perspective transformation is a geometric operation that maps a 2D image from one plane to another, simulating the effect of viewing an image from a different angle. It is commonly used in tasks like image rectification (e.g., correcting the perspective of a document image taken at an angle) and creating panoramic views. The transformation is defined by four pairs of points: one set from the source image and one set from the destination image. The operation finds a transformation matrix, which is then applied to warp the source image to the new perspective.

Source Code:

```
 # Define four points in the original image
pts_before = np.float32([[50, 50], [200, 50], [50, 200], [200, 200]])

# Define where these points should map in the output image (change of perspective)
pts_after = np.float32([[10, 100], [180, 50], [100, 250], [220, 220]])

# Get the perspective transformation matrix
perspective_matrix = cv2.getPerspectiveTransform(pts_before, pts_after)

# Apply the perspective transformation
warped_image = cv2.warpPerspective(image_rgb, perspective_matrix, (w, h))

# Display the warped image
plt.imshow(warped_image)
plt.title('Perspective Transformed Image')
plt.axis('off')
plt.show()
```

Output:



Viva Questions

1. What is perspective transformation, and how does it differ from affine transformation?
   - This tests your understanding of how perspective transformation alters the view of an image, including depth perception, compared to affine transformation, which maintains parallelism.

2. Can you explain the role of homography in perspective transformation?
   - This question assesses your knowledge of homography, the matrix used to map points between different perspectives and how it is computed.

3. What are the key applications of perspective transformation in real-world scenarios?
   - This focuses on your ability to connect theory with practice by discussing use cases like document scanning, 3D mapping, and correcting image distortions.

4. How do you determine the four points for perspective transformation, and what challenges might arise in choosing them?
   - This probes your practical understanding of selecting source and destination points, as well as issues like point misalignment and how they affect the transformation.

5. What is the role of interpolation in perspective transformation, and how do different interpolation methods affect the output?
   - This explores your knowledge of how interpolation (e.g., nearest neighbor, bilinear) is used during transformations and how it affects the accuracy and quality of the transformed image.

Experiment-8

Aim: Implementation of Camera
Calibration

Theory: Camera calibration is the process of estimating the internal and external parameters of a camera to correct lens distortion and determine the relationship between the 3D real-world coordinates and the 2D coordinates of the image. This is important for applications such as 3D reconstruction, robot navigation, and augmented reality. Calibration typically involves capturing images of a known reference object (such as a chessboard) and finding correspondences between the object's 3D points and the image's 2D points.

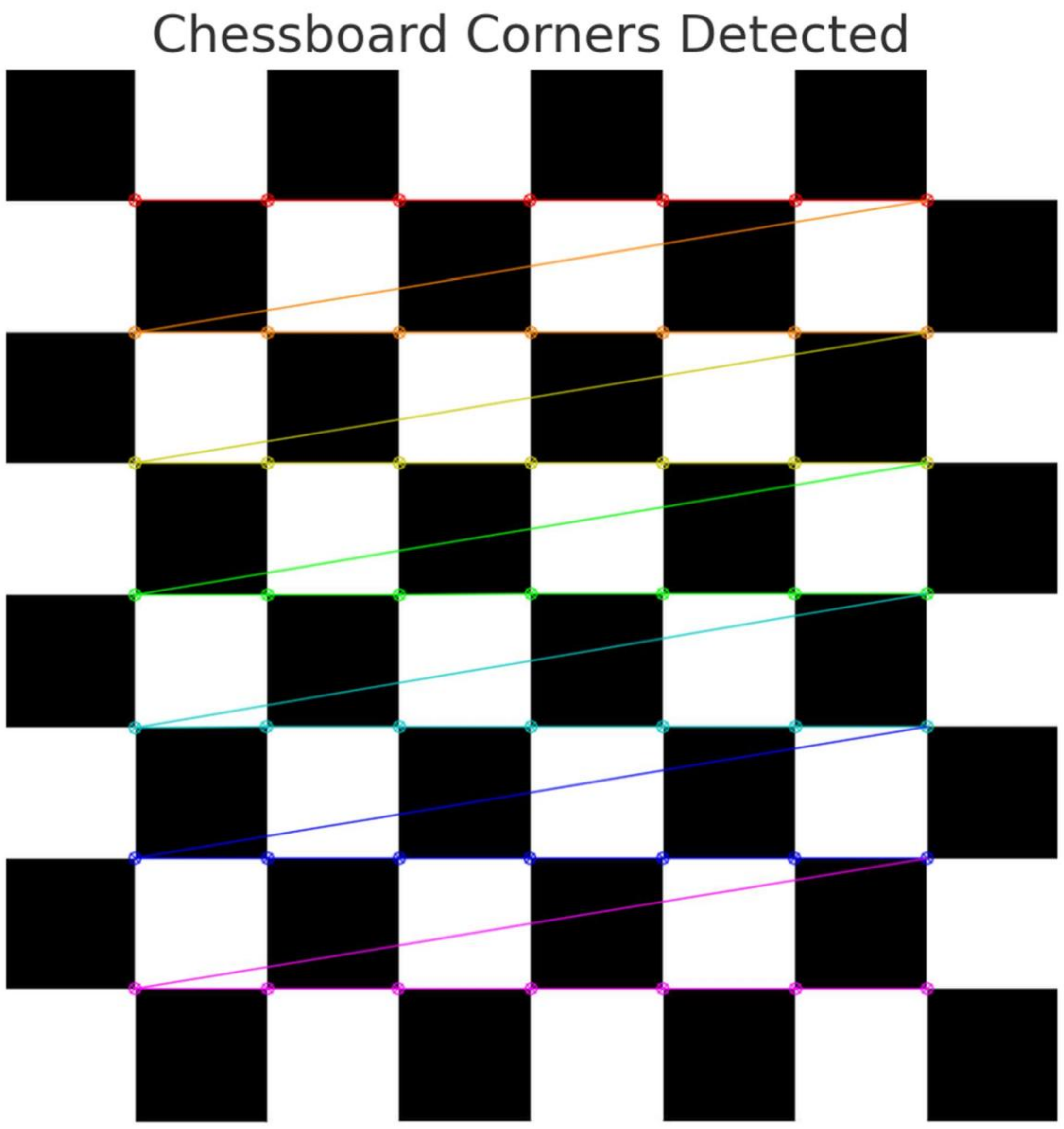
Steps in camera calibration:

1. Capture multiple images of a calibration pattern (like a chessboard).
2. Detect the pattern in the images and extract corner points.
3. Use these correspondences to compute the camera matrix and distortion coefficients.

Source Code:

```
 import cv2
import numpy as np

# Load the image of the chessboard (for example)
image = cv2.imread(/content.chess.jpg)

# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Define the size of the chessboard pattern (number of internal corners)
pattern_size = (9, 6)  # Example for 9x6 chessboard

# Find the chessboard corners
ret, corners = cv2.findChessboardCorners(gray, pattern_size, None)

# If corners are found, proceed with calibration
if ret:
   # Draw the corners on the image
   image_with_corners = cv2.drawChessboardCorners(image, pattern_size, corners, ret)

   # Display the image with detected corners
   plt.imshow(cv2.cvtColor(image_with_corners, cv2.COLOR_BGR2RGB))
   plt.title('Chessboard Corners Detected')
   plt.axis('off')
   plt.show()

# In a full calibration process, multiple images would be used to compute
# the camera matrix and distortion coefficients.
```

Output:



Viva Questions

1. What is camera calibration, and why is it essential in computer vision?
   - This question assesses your understanding of camera calibration, focusing on its role in correcting lens distortion and improving the accuracy of 3D measurements.

2. Can you explain the difference between intrinsic and extrinsic parameters in camera calibration?
   - This probes your knowledge of the two main types of camera parameters, where intrinsic parameters relate to the camera's internal characteristics, while extrinsic parameters describe its position and orientation in the world.

3. What techniques are commonly used for camera calibration, and how do they work?
   - This question evaluates your familiarity with methods such as Zhang's method using a checkerboard pattern, point correspondences, and their algorithms for estimating camera parameters.

4. How do you assess the accuracy of camera calibration, and what metrics do you use?
   - This focuses on your understanding of evaluating calibration results, including the use of reprojection error and how it helps in determining calibration quality.

5. What challenges might arise during the camera calibration process, and how can they be mitigated?
   - This question looks for insights into practical difficulties, such as noise, insufficient data points, or incorrect assumptions, and strategies to improve calibration robustness.

Experiment-9

Aim: Compute
Fundamental Matrix

Theory: The fundamental matrix FFF relates corresponding points between two stereo images. It encapsulates the epipolar geometry, ensuring that if a point is visible in one image, its corresponding point in the other image lies along a specific line called the epipolar line. The fundamental matrix is critical in applications like 3D reconstruction, depth estimation, and stereo vision. It is a 3x3 matrix derived from corresponding points in both images and is used to calculate the epipolar constraint:

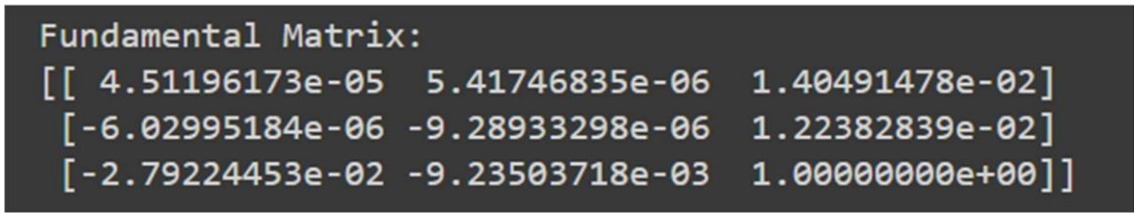
$$x'^T F x = 0$$

where x and x′ are corresponding points in the two images.

Source Code:

```
 import cv2
import numpy as np

pts_image1 = np.array([[100, 150], [120, 200], [130, 220], [150, 180],

                  [160, 140], [170, 190], [180, 210], [190, 160]], dtype=np.float32)

pts_image2 = np.array([[90, 140], [115, 190], [125, 215], [145, 175],

                  [155, 135], [165, 185], [175, 205], [185, 155]], dtype=np.float32)

# Compute the fundamental matrix

F, mask = cv2.findFundamentalMat(pts_image1, pts_image2, cv2.FM_LMEDS)

if F is not None:

   print("Fundamental Matrix:")

   print(F)

else:

   print("Fundamental Matrix computation failed.")
```

Output:



```
Fundamental Matrix:
[[ 4.51196173e-05  5.41746835e-06  1.40491478e-02]
 [-6.02995184e-06 -9.28933298e-06  1.22382839e-02]
 [-2.79224453e-02 -9.23503718e-03  1.00000000e+00]]
```

Viva Questions

1. What is the fundamental matrix in computer vision, and why is it important?
   - This question assesses your understanding of the fundamental matrix, emphasizing its role in relating corresponding points between two images in stereo vision.

2. Can you explain the relationship between the fundamental matrix and epipolar geometry?
   - This probes your knowledge of how the fundamental matrix encodes the geometric relationships between two views of the same scene, specifically in the context of epipolar lines.

3. What are the key steps involved in computing the fundamental matrix from point correspondences?
   - This question evaluates your understanding of the process, including the collection of point correspondences, normalization, solving the homogeneous system, and enforcing the rank-2 constraint.

4. How do you handle noise and outliers when computing the fundamental matrix?
   - This focuses on your knowledge of robust estimation techniques, such as RANSAC, and how they help improve the accuracy of the fundamental matrix estimation in real-world scenarios.

5. What metrics or methods can be used to evaluate the accuracy of the computed fundamental matrix?
   - This question looks for insights into validation methods, such as reprojection error, and how to assess the consistency of the epipolar geometry.