**Pravakar Das**

**ID:24241266**

**Task 1:**

import numpy as np

import matplotlib.pyplot as plt

#mycode="PR66" cause Name:PRAVAKAR ID:24241266

P\_dzn = np.array([

[0,0, 0 , 0 , 0 , 0 , 0,0],

[0,0, 0 , 0 , 0 , 0 , 0,0],

[0,0, 255, 255, 255, 255, 0,0],

[0,0, 255, 0 , 0 , 255, 0,0],

[0,0, 255, 255, 255, 255, 0,0],

[0,0, 255, 0 , 0 , 0 , 0,0],

[0,0, 255, 0 , 0 , 0 , 0,0],

[0,0, 0 , 0 , 0 , 0 , 0,0],

[0,0, 0 , 0 , 0 , 0 , 0,0]

])

R\_dzn = np.array([

[0,0, 0 , 0 , 0 , 0 , 0,0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 255, 255, 255, 255, 0, 0],

[0, 0, 255, 0, 0, 255, 0, 0],

[0, 0, 255, 255, 255, 255, 0, 0],

[0, 0, 255, 0, 255, 0, 0, 0],

[0, 0, 255, 0, 0, 255, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0,0, 0 , 0 , 0 , 0 , 0,0]

])

num6\_dzn = np.array([

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 255, 255, 255, 255, 0, 0],

[0, 0, 255, 0, 0, 0, 0, 0],

[0, 0, 255, 255, 255, 255, 0, 0],

[0, 0, 255, 0, 0, 255, 0, 0],

[0, 0, 255, 255, 255, 255, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0],

])

result=np.hstack([P\_dzn,R\_dzn,num6\_dzn,num6\_dzn])

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

plt.axis('off')

plt.show()

****

**Task 2:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

import os

def crop\_image(img, start\_x, start\_y, end\_x, end\_y):

# This is useful to focus on a specific area of the image by removing irrelevant parts, which can help in reducing the size of the image and focusing on key features.

return img[start\_y:end\_y, start\_x:end\_x]

def flip\_image(img, flip\_type):

# Flipping is a common operation in data augmentation to create variations of the image, ensuring the model becomes more robust to different orientations of objects.

return cv2.flip(img, flip\_type)

def rotate\_image(img, angle, scale=1.0):

# This transformation simulates the effect of objects being captured from different angles which helps improve the robustness of a model by introducing rotational variance to the training data.

rows, cols = img.shape[:2]

rotation\_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, scale)

return cv2.warpAffine(img, rotation\_matrix, (cols, rows))

def resize\_image(img, width, height):

# By doing this, standardize all images to a consistent size, regardless of their original dimensions.

dim = (width, height)

return cv2.resize(img, dim, interpolation=cv2.INTER\_AREA)

def translate\_image(img, shift\_x, shift\_y):

# This transformation simulates small movements of the camera or object in the scene, making the model more resilient to such shifts during real-world use.

translation\_matrix = np.float32([[1, 0, shift\_x], [0, 1, shift\_y]])

return cv2.warpAffine(img, translation\_matrix, (img.shape[1], img.shape[0]))

def shear\_image(img, shear\_x, shear\_y):

# Shearing distorts the shape of the image in a controlled manner. This simulates the effect of objects being captured from an angle or a perspective view and increases model robustness to such distortions.

rows, cols = img.shape[:2]

shear\_matrix = np.float32([[1, shear\_x, 0], [shear\_y, 1, 0]])

return cv2.warpAffine(img, shear\_matrix, (cols, rows))

def process\_and\_save\_images(source\_dir, destination\_dir, resize\_dimen):

for filename in os.listdir(source\_dir):

if filename.endswith((".png", ".jpg", ".jpeg")):

img\_path = os.path.join(source\_dir, filename)

img = cv2.imread(img\_path)

if img is None:

print(f"Failed to load image: {img\_path}")

continue

cropped = crop\_image(img, 100, 100, 277, 243)

flipped = flip\_image(img, 0)

rotated = rotate\_image(img, -90, 0.5)

resized = resize\_image(img, resize\_dimen[0], resize\_dimen[1])

translated = translate\_image(img, 50, 30)

sheared = shear\_image(img, 0.2, 0.0)

base\_name, ext = os.path.splitext(filename)

cropped\_filename = f"cropped\_{base\_name}{ext}"

cropped\_destination\_path = os.path.join(destination\_dir, cropped\_filename)

plt.imsave(cropped\_destination\_path, cropped)

flipped\_filename = f"flipped\_{base\_name}{ext}"

flipped\_destination\_path = os.path.join(destination\_dir, flipped\_filename)

plt.imsave(flipped\_destination\_path, flipped)

rotated\_filename = f"rotated\_{base\_name}{ext}"

rotated\_destination\_path = os.path.join(destination\_dir, rotated\_filename)

plt.imsave(rotated\_destination\_path, rotated)

resized\_filename = f"resized\_{base\_name}{ext}"

resized\_destination\_path = os.path.join(destination\_dir, resized\_filename)

plt.imsave(resized\_destination\_path, resized)

translated\_filename = f"translated\_{base\_name}{ext}"

translated\_destination\_path = os.path.join(destination\_dir, translated\_filename)

plt.imsave(translated\_destination\_path, translated)

sheared\_filename = f"sheared\_{base\_name}{ext}"

sheared\_destination\_path = os.path.join(destination\_dir, sheared\_filename)

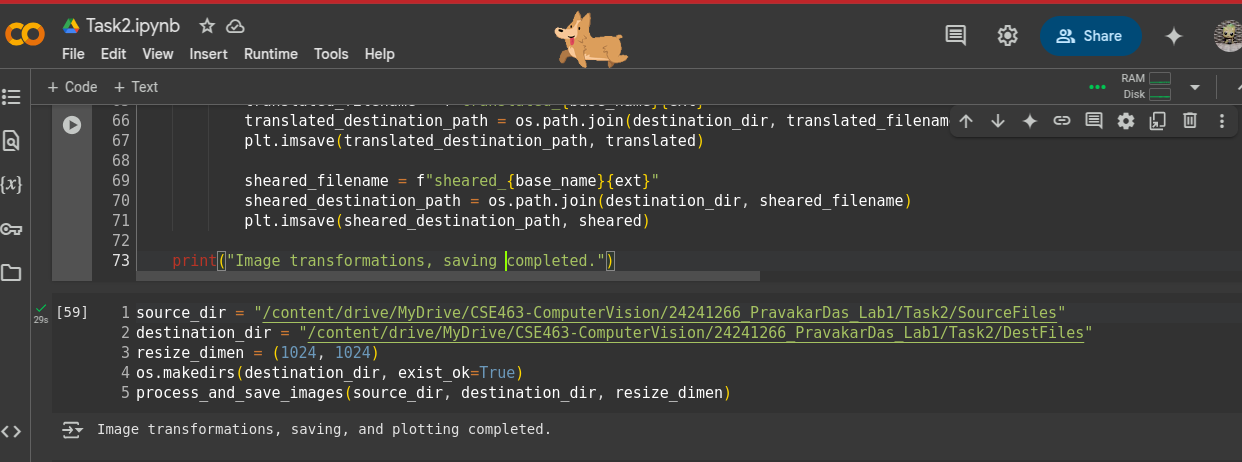
plt.imsave(sheared\_destination\_path, sheared)  
source\_dir = "/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task2/SourceFiles"

destination\_dir = "/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task2/DestFiles"

resize\_dimen = (1024, 1024)

os.makedirs(destination\_dir, exist\_ok=True)

process\_and\_save\_images(source\_dir, destination\_dir, resize\_dimen)



===================

def salt\_and\_pepper\_noise(image, prob):

noisy\_image = np.copy(image)

num\_salt = np.ceil(prob \* image.size \* 0.5).astype(int) # Ensure this is an integer

num\_pepper = np.ceil(prob \* image.size \* 0.5).astype(int) # Ensure this is an integer

salt\_coords = [np.random.randint(0, i - 1, num\_salt) for i in image.shape]

noisy\_image[salt\_coords[0], salt\_coords[1]] = 255

pepper\_coords = [np.random.randint(0, i - 1, num\_pepper) for i in image.shape]

noisy\_image[pepper\_coords[0], pepper\_coords[1]] = 0

return noisy\_image

def gaussian\_noise(image, mean=0, std=50):

noise = np.random.normal(mean, std, image.shape)

noisy\_image = np.clip(image + noise, 0, 255).astype(np.uint8)

return noisy\_image

def apply\_noise(image, prob\_salt\_pepper=0.05, gaussian\_std=50):

salt\_pepper = salt\_and\_pepper\_noise(image, prob\_salt\_pepper)

gaussian = gaussian\_noise(image, std=gaussian\_std)

return salt\_pepper, gaussian

def plot\_histograms\_side\_by\_side(salt\_pepper, gaussian, filename):

plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 1)

plt.hist(salt\_pepper.flatten(), bins=50, density=True, alpha=0.7, color='r')

plt.title(f'Salt-and-Pepper Histogram - {filename}')

plt.xlabel('Pixel Value')

plt.ylabel('Frequency')

plt.subplot(1, 2, 2)

plt.hist(gaussian.flatten(), bins=50, density=True, alpha=0.7, color='g')

plt.title(f'Gaussian Noise Histogram - {filename}')

plt.xlabel('Pixel Value')

plt.ylabel('Frequency')

plt.tight\_layout()

plt.show()

def plot\_noisy\_images\_with\_histograms(original, salt\_pepper, gaussian, filename):

fig, axes = plt.subplots(1, 3, figsize=(20, 10))

axes[0].set\_title(f'Original - {filename}')

axes[0].imshow(cv2.cvtColor(original, cv2.COLOR\_BGR2RGB))

axes[0].axis('off')

axes[1].set\_title(f'Salt-and-Pepper Noise - {filename}')

axes[1].imshow(cv2.cvtColor(salt\_pepper, cv2.COLOR\_BGR2RGB))

axes[1].axis('off')

axes[2].set\_title(f'Gaussian Noise - {filename}')

axes[2].imshow(cv2.cvtColor(gaussian, cv2.COLOR\_BGR2RGB))

axes[2].axis('off')

plt.tight\_layout()

plt.show()

plot\_histograms\_side\_by\_side(salt\_pepper, gaussian, filename)

def process\_images(source\_dir, destination\_dir, resize\_dimen):

for filename in os.listdir(source\_dir):

if filename.endswith((".png", ".jpg", ".jpeg")):

img\_path = os.path.join(source\_dir, filename)

img = cv2.imread(img\_path)

if img is None:

print(f"Failed to load image: {img\_path}")

continue

img\_resized = cv2.resize(img, resize\_dimen)

salt\_pepper, gaussian = apply\_noise(img\_resized)

base\_name, ext = os.path.splitext(filename)

salt\_pepper\_filename = f"salt\_pepper\_{base\_name}{ext}"

salt\_pepper\_path = os.path.join(destination\_dir, salt\_pepper\_filename)

plt.imsave(salt\_pepper\_path, cv2.cvtColor(salt\_pepper, cv2.COLOR\_BGR2RGB))

gaussian\_filename = f"gaussian\_{base\_name}{ext}"

gaussian\_path = os.path.join(destination\_dir, gaussian\_filename)

plt.imsave(gaussian\_path, cv2.cvtColor(gaussian, cv2.COLOR\_BGR2RGB))

plot\_noisy\_images\_with\_histograms(img\_resized, salt\_pepper, gaussian, filename)

print("Image noise transformation, saving, plotting, and histogram display completed.")

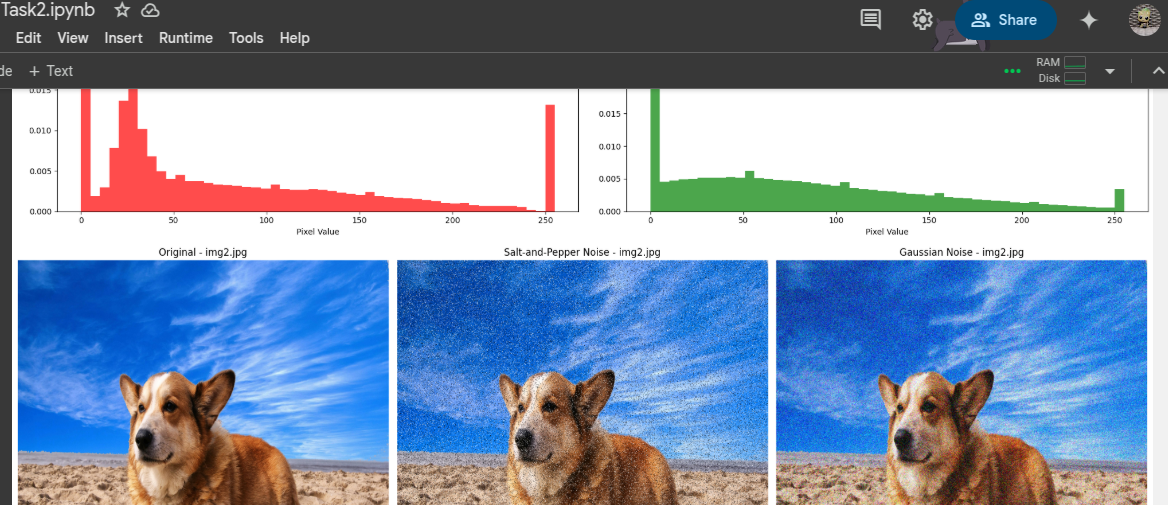
source\_dir = "/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task2/SourceFiles"

destination\_dir = "/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task2/DestFiles"

resize\_dimen = (1024, 1024)

os.makedirs(destination\_dir, exist\_ok=True)

process\_images(source\_dir, destination\_dir, resize\_dimen)



**Task 3:**

import cv2

import os

import numpy as np

import matplotlib.pyplot as plt

def load\_and\_convert\_images(source\_dir):

image\_filenames = [f for f in os.listdir(source\_dir) if f.endswith(('.jpg', '.jpeg', '.png'))]

images\_rgb = []

for filename in image\_filenames:

img\_bgr = cv2.imread(os.path.join(source\_dir, filename))

img\_rgb = cv2.cvtColor(img\_bgr, cv2.COLOR\_BGR2RGB)

images\_rgb.append(img\_rgb)

return images\_rgb

def resize\_images(images\_rgb, width=2048, height=2048):

images\_resized = []

for img in images\_rgb:

dim = (width, height)

img\_resized = cv2.resize(img, dim, interpolation=cv2.INTER\_AREA)

images\_resized.append(np.uint8(img\_resized))

return images\_resized

def blend\_images(images\_resized, alpha\_values=None):

if alpha\_values is None:

alpha\_values = [0.2] \* len(images\_resized)

blended\_image = images\_resized[0] \* alpha\_values[0]

for i in range(1, len(images\_resized)):

blended\_image = np.uint8(blended\_image)

current\_image = np.uint8(images\_resized[i])

blended\_image = cv2.addWeighted(blended\_image, 1 - alpha\_values[i], current\_image, alpha\_values[i], 0)

plt.imshow(blended\_image)

plt.title("Blended Image")

plt.show()

return blended\_image

def save\_image(image, output\_path):

cv2.imwrite(output\_path, cv2.cvtColor(image, cv2.COLOR\_RGB2BGR))

source\_dir = "/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task3/SourceFiles"

destination\_dir = "/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task3/DestFiles"

images\_rgb = load\_and\_convert\_images(source\_dir)

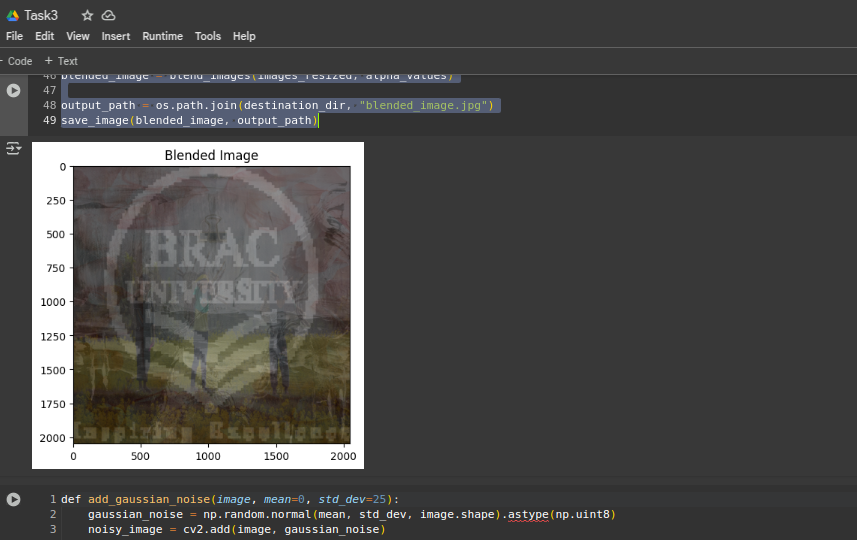
images\_resized = resize\_images(images\_rgb)

alpha\_values = [0.25, 0.25, 0.2, 0.15, 0.15]

blended\_image = blend\_images(images\_resized, alpha\_values)

output\_path = os.path.join(destination\_dir, "blended\_image.jpg")

save\_image(blended\_image, output\_path)

===========================================

def add\_gaussian\_noise(image, mean=0, std\_dev=25):

gaussian\_noise = np.random.normal(mean, std\_dev, image.shape).astype(np.uint8)

noisy\_image = cv2.add(image, gaussian\_noise)

noisy\_image = np.clip(noisy\_image, 0, 255)

return noisy\_image

def display\_before\_after(original\_image, noisy\_image):

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

plt.subplot(1, 2, 1)

plt.imshow(original\_image)

plt.title("Original Image")

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(noisy\_image)

plt.title("Image with Gaussian Noise")

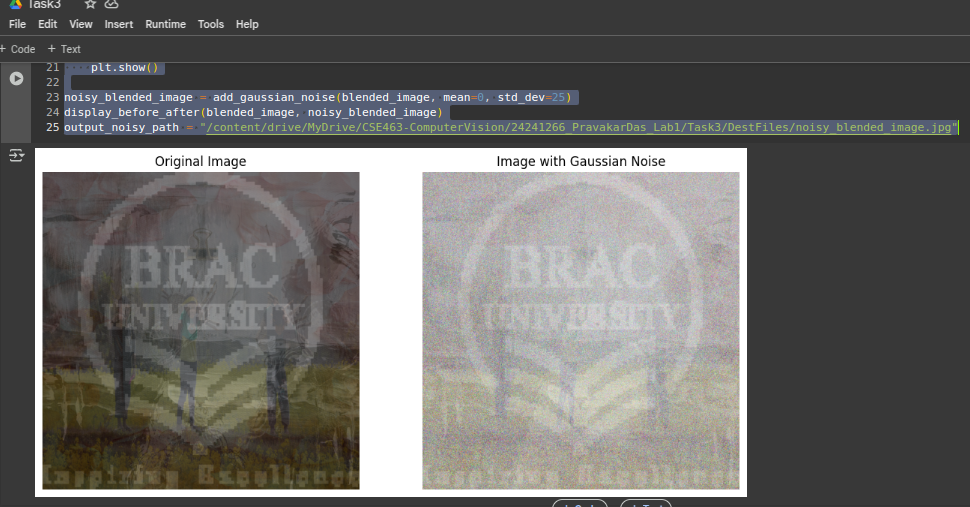
plt.axis('off')

plt.show()

noisy\_blended\_image = add\_gaussian\_noise(blended\_image, mean=0, std\_dev=25)

display\_before\_after(blended\_image, noisy\_blended\_image)

output\_noisy\_path = "/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task3/DestFiles/noisy\_blended\_image.jpg"



==================================================

mean = 0

std\_dev = 1

num\_samples = 100000

gaussian\_noise = np.random.normal(mean, std\_dev, num\_samples)

plt.hist(gaussian\_noise, bins=50, density=True, alpha=0.6, color='g')

plt.title('Generated Gaussian Noise')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

image = cv2.imread('/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab1/Task3/DestFiles/noisy\_blended\_image.jpg')

