

Computer Vision

Course Code: MEAD-652

Lab Practical File

Submitted by

Shantanu Shukla

01211805424

**in partial fulfillment for the award of the degree
of**

Master of Technology (AI & DS)

batch of 2024 - 26 (2nd Semester)



Center for Development of Advanced Computing, Noida

Affiliated to Guru Gobind Singh Indraprastha University

| S. No | Title | Page No. | Date | Signature |
|-------|---|----------|------------|-----------|
| 1 | Exercise-1: i. Matrix operations using NumPy ii. Basic image manipulations iii. Singular Value Decomposition | 3-8 | 27-01-2025 | |
| 2 | Exercise-2: Sobel Edge Detection | 9 | 03-02-2025 | |
| 3 | Exercise-3: Harris Corner Detection | 10 | 10-02-2025 | |
| 4 | Exercise-4: 2D - Transformations on an image: i. Shear ii. Scale iii. Transition iv. Rotation v. Affine | 11-13 | 17-02-2025 | |
| 5 | Exercise-5: Box Filter | 14 | 03-03-2025 | |
| 6 | Exercise-6: Shadow Effect | 15-16 | 10-03-2025 | |
| 7 | Exercise-7: Gaussian Filtering | 17 | 17-03-2025 | |
| 8 | Exercise-8: Laplacian Filtering | 18 | 24-03-2025 | |
| 9 | Exercise-9: Gaussian Pyramid | 19 | 07-04-2025 | |
| 10 | Exercise-10: Image Classification using CNN | 20-33 | 21-04-2025 | |

Exercise-1

i. Matrix Operations using Numpy:

Code:

```
1. import numpy as np
2.
3. M = np.array(
4.     [
5.         [1, 2, 3],
6.         [4, 5, 6],
7.         [7, 8, 9],
8.         [0, 2, 2],
9.     ]
10. )
11.
12. a = np.array(
13.     [
14.         [1],
15.         [1],
16.         [0],
17.     ]
18. )
19.
20. b = np.array(
21.     [
22.         [-1],
23.         [2],
24.         [5],
25.     ]
26. )
27.
28. c = np.array(
29.     [
30.         [0],
31.         [2],
32.         [3],
33.         [2],
34.     ]
35. )
36.
37.
38.
39. print(np.dot(a.T, b))
40. print(np.multiply(a, b).T)
41. print(np.multiply(a, b).T)
42. print(M * a.T)
43. print(np.sort(M))
```

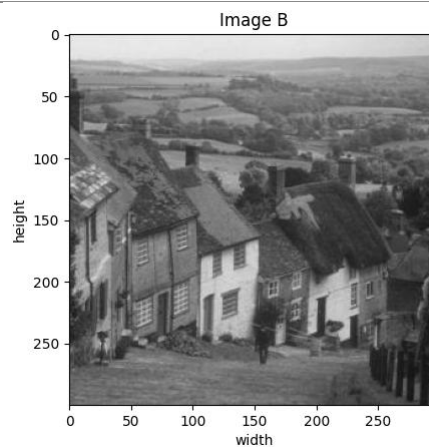
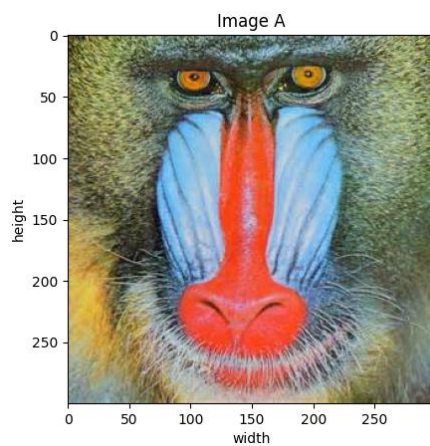
Output:

```
• (CV) vscode → /workspaces/cdac-labwork/sem-2/CV (main) $ python exercise-1/1.py
[[1]]
[[-1  2  0]]
[[-1  2  0]]
[[1  2  0]
 [4  5  0]
 [7  8  0]
 [0  2  0]]
[[1  2  3]
 [4  5  6]
 [7  8  9]
 [0  2  2]]
○ (CV) vscode → /workspaces/cdac-labwork/sem-2/CV (main) $
```

ii. Basic Image Manipulations:

Code:

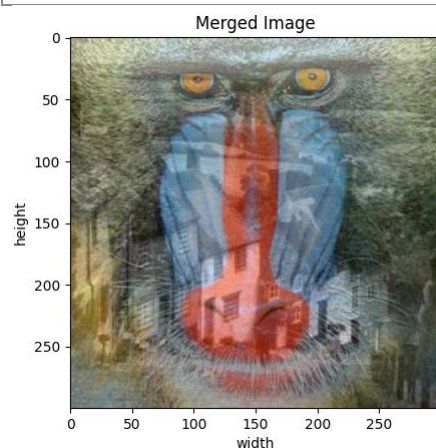
```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. import skimage as sk
4.
5. imageA = sk.io.imread('./image1.jpg')
6. imageB = sk.io.imread('./image2.jpg')
7.
8. plt.figure()
9. plt.title('Image A')
10. plt.xlabel('width')
11. plt.ylabel('height')
12. plt.imshow(imageA)
13.
14. plt.figure()
15. plt.title('Image B')
16. plt.xlabel('width')
17. plt.ylabel('height')
18. plt.imshow(imageB)
```



```

1. # Convert the images to double precision
2. imageA = imageA.astype(np.float64)
3. imageB = imageB.astype(np.float64)
4.
5. # rescale them to stretch from minimum value 0 to maximum value 1
6. imageA = (imageA - np.min(imageA)) / (np.max(imageA) - np.min(imageA))
7. imageB = (imageB - np.min(imageB)) / (np.max(imageB) - np.min(imageB))
8.
9. merged_image = imageA + imageB
10. merged_image = (merged_image - np.min(merged_image)) / (np.max(merged_image) -
np.min(merged_image))
11.
12. # set size of the result image
13. merged_image = (merged_image * 255).astype(np.uint8)
14.
15. # print merged image
16. plt.figure()
17. plt.title('Merged Image')
18. plt.xlabel('width')
19. plt.ylabel('height')
20. plt.imshow(merged_image)
21.
22. # save merged image
23. sk.io.imshow('./merged.jpg', merged_image)

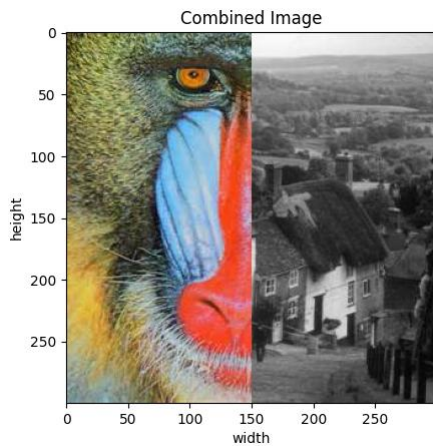
```



```

1. height, width, channels = imageA.shape
2. combined_image = np.zeros_like(imageA)
3.
4. combined_image[:, :width//2, :] = imageA[:, :width//2, :]
5. combined_image[:, width//2:, :] = imageB[:, width//2:, :]
6.
7. # print combined image
8. plt.figure()
9. plt.title('Combined Image')
10. plt.xlabel('width')
11. plt.ylabel('height')
12. plt.imshow(combined_image)
13.
14. # set size of the result image
15. combined_image = (combined_image * 255).astype(np.uint8)
16.
17. # save combined image
18. sk.io.imshow('./combined.jpg', combined_image)

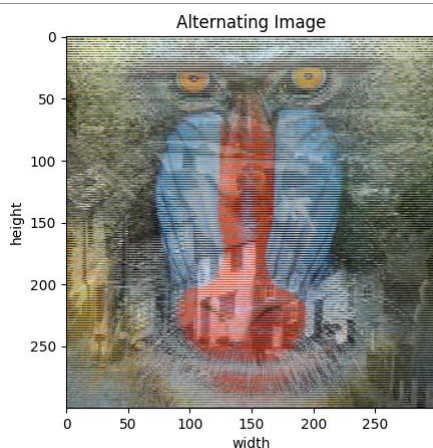
```



```

1. alternating_image = np.zeros_like(imageA)
2. for i in range(height):
3.     if i % 2 == 0:
4.         alternating_image[i, :, :] = imageA[i, :, :]
5.     else:
6.         alternating_image[i, :, :] = imageB[i, :, :]
7.
8. # set size of the result image
9. alternating_image = (alternating_image * 255).astype(np.uint8)
10.
11. # print alternating image
12. plt.figure()
13. plt.title('Alternating Image')
14. plt.xlabel('width')
15. plt.ylabel('height')
16. plt.imshow(alternating_image)
17.
18. sk.io.imsave('alternating_image.jpg', alternating_image)

```



```

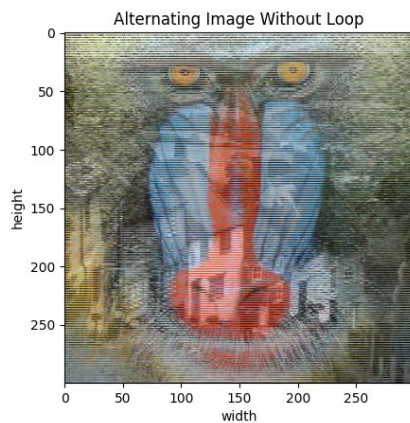
1. rows_imageA = imageA[::2, :, :]
2. rows_imageB = imageB[1::2, :, :]
3.
4. alternating_image_without_loop = np.zeros_like(imageA)
5. alternating_image_without_loop[::2, :, :] = rows_imageA
6. alternating_image_without_loop[1::2, :, :] = rows_imageB

```

```

1. # print alternating image
2. plt.figure()
3. plt.title('Alternating Image Without Loop')
4. plt.xlabel('width')
5. plt.ylabel('height')
6. plt.imshow(alternating_image_without_loop)
7.
8. # set size of the result image
9. alternating_image_without_loop = (alternating_image_without_loop *
255).astype(np.uint8)
10. sk.io.imshow('alternating_image_without_loop.jpg',
alternating_image_without_loop)

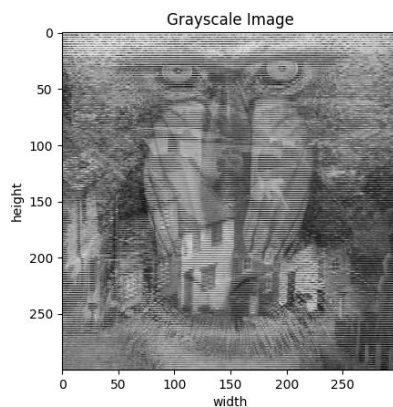
```



```

1. grayscale_image = sk.color.rgb2gray(alternating_image_without_loop)
2.
3. # set size of the result image
4. grayscale_image = (grayscale_image * 255).astype(np.uint8)
5.
6. # print grayscale image
7. plt.figure()
8. plt.title('Grayscale Image')
9. plt.xlabel('width')
10. plt.ylabel('height')
11. plt.imshow(grayscale_image, cmap='gray')
12. sk.io.imshow('grayscale_image.png', grayscale_image)

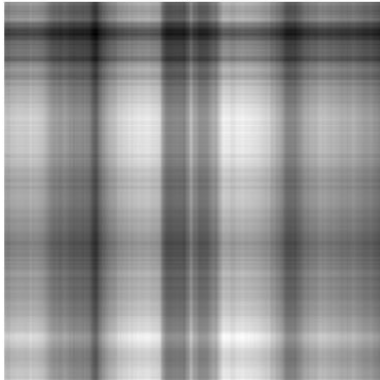
```



iii. Singular Value Decomposition

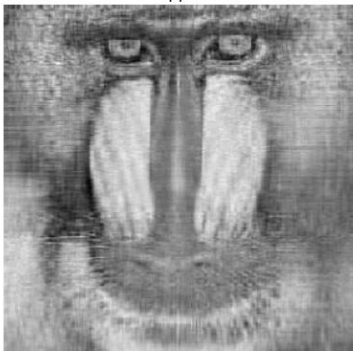
```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. from skimage import io
4.
5. imageA_grey = io.imread('./image1.jpg', as_gray=True)
6.
7. U, S, Vt = np.linalg.svd(imageA_grey, full_matrices=False)
8. rank = 1
9. rank_1_approx = U[:, :rank] @ np.diag(S[:rank]) @ Vt[:rank, :]
10.
11. # Save and display the rank-1 approximation
12. plt.imshow(rank_1_approx, cmap='gray')
13. plt.title(f'Rank-{rank} Approximation')
14. plt.axis('off')
15. plt.savefig('rank_1_approx.jpg', bbox_inches='tight', pad_inches=0)
16. plt.show()
```

Rank-1 Approximation



```
1. rank = 20
2. rank_20_approx = U[:, :rank] @ np.diag(S[:rank]) @ Vt[:rank, :]
3.
4. # Save and display the rank-20 approximation
5. plt.imshow(rank_20_approx, cmap='gray')
6. plt.title(f'Rank-{rank} Approximation')
7. plt.axis('off')
8. plt.savefig('rank_20_approx.png', bbox_inches='tight', pad_inches=0)
9. plt.show()
```

Rank-20 Approximation



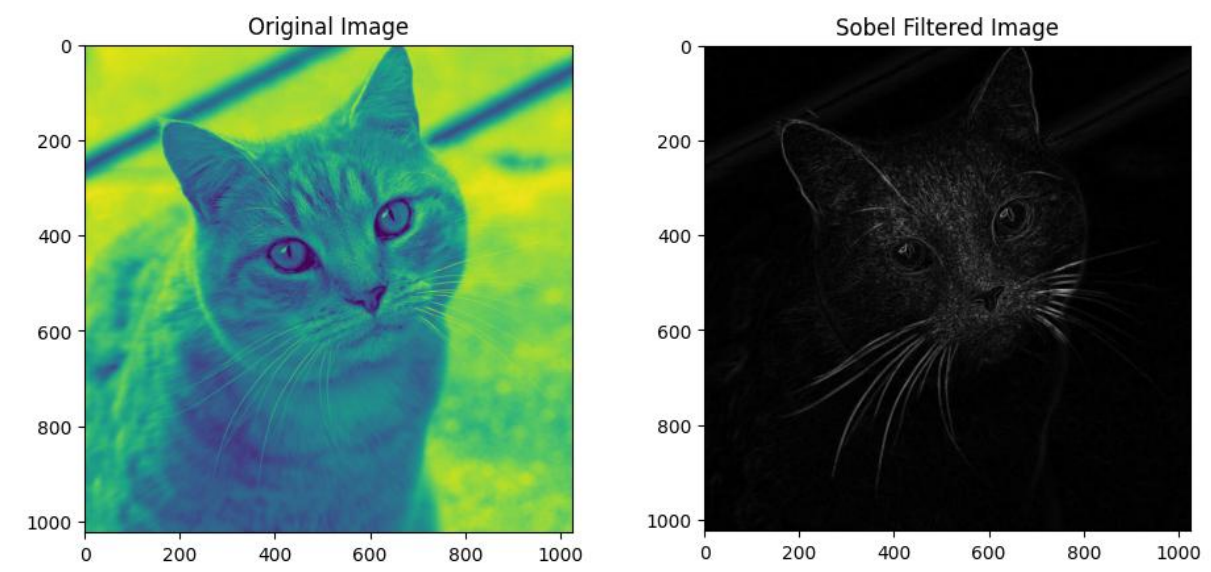
Exercise-2

Objective: Sobel Edge Detection

Code:

```
1. import skimage as sk
2. import matplotlib.pyplot as plt
3.
4. image = sk.io.imread("./image.png", sk.color.rgb2gray)
5. plt.imshow(image)
6. plt.title('Original Image')
7. plt.show()
8.
9.
10. edges = sk.filters.sobel(image)
11.
12. plt.imshow(edges, cmap='gray')
13. plt.title('Sobel Filtered Image')
14. plt.show()
```

Output:



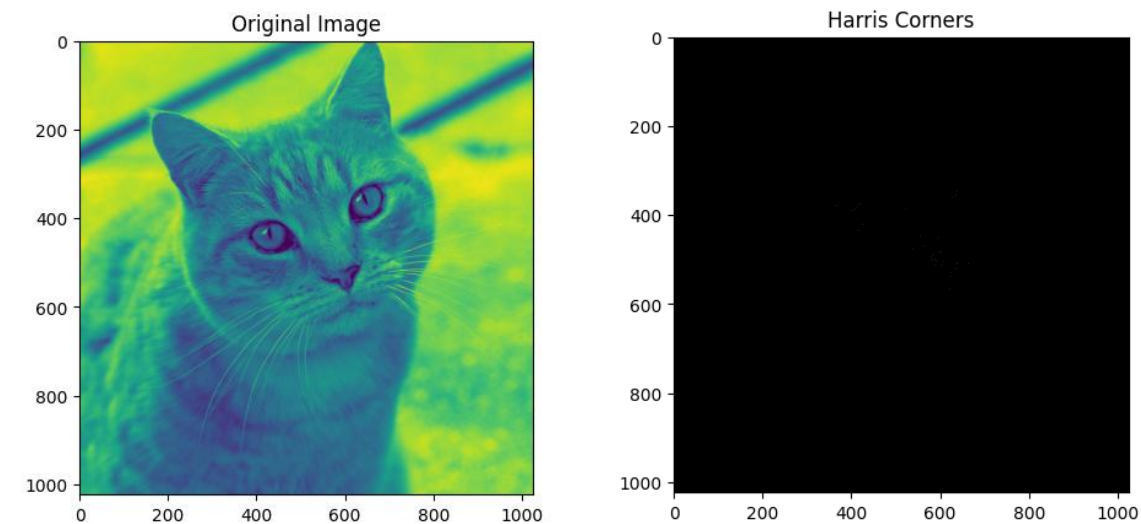
Exercise-3

Objective: Harris Corner Detection

Code:

```
1. import skimage as sk
2. import matplotlib.pyplot as plt
3. import numpy as np
4.
5. image = sk.io.imread('./image.png', sk.color.rgb2gray)
6. plt.imshow(image)
7. plt.title('Original Image')
8. plt.show()
9.
10. harris = sk.feature.corner_harris(image)
11. corners = sk.feature.corner_peaks(harris, min_distance=2, threshold_rel=0.02)
12.
13. corner_image = np.zeros_like(image)
14. corner_image[corners[:, 0], corners[:, 1]] = 1
15.
16. plt.imshow(corner_image, cmap='gray')
17. plt.title('Harris Corners')
18. plt.show()
```

Ouput:

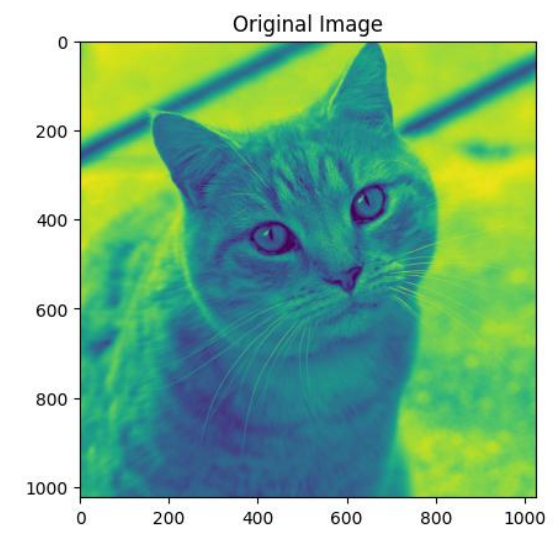


Exercise-4

Objective: 2D - Transformations on an image

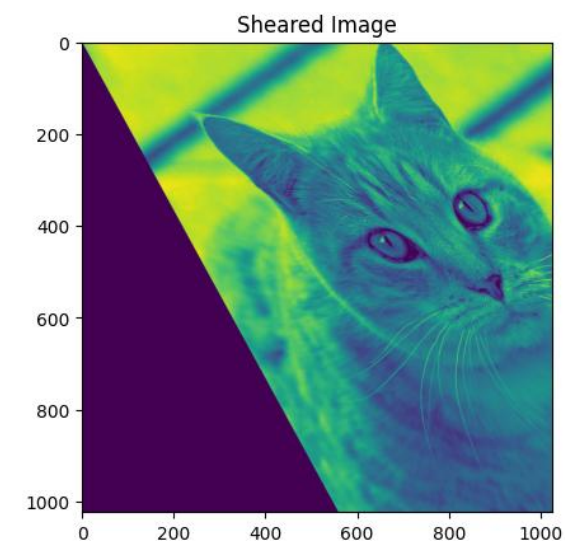
Code:

```
1. import skimage as sk
2. import matplotlib.pyplot as plt
3. image = sk.io.imread("../image.png", as_gray=True)
4. plt.title("Original Image")
5. plt.imshow(image)
```



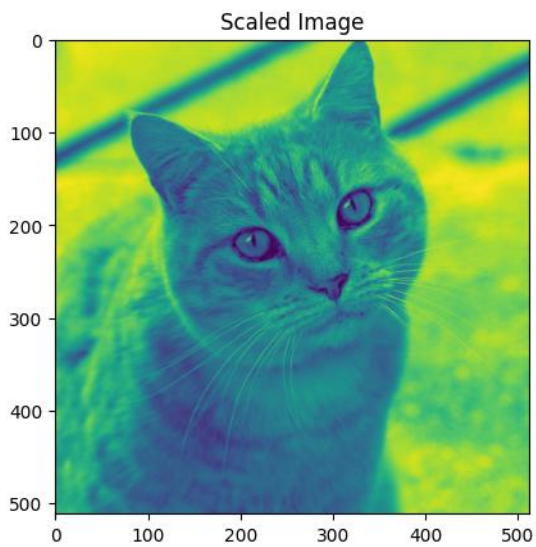
1. Shear

```
1. shear = sk.transform.warp(image, sk.transform.AffineTransform(shear=0.5))
2. plt.title("Sheared Image")
3. plt.imshow(shear)
```



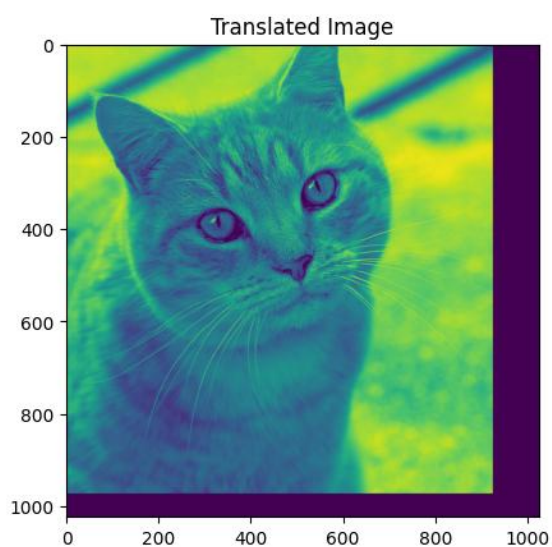
2. Scale

```
1. scaled = sk.transform.rescale(image, 0.5)
2. plt.title("Scaled Image")
3. plt.imshow(scaled)
4. plt.show()
```



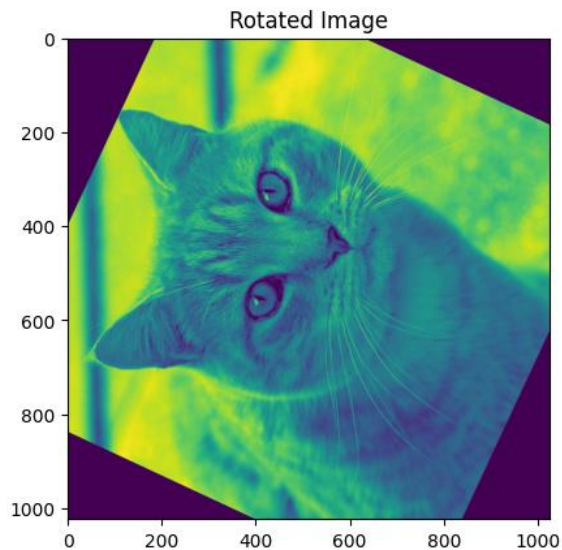
3. Transition

```
1. transition = sk.transform.warp(image,
sk.transform.AffineTransform(translation=(100, 50)))
2. plt.title("Translated Image")
3. plt.imshow(transition)
4. plt.show()
```



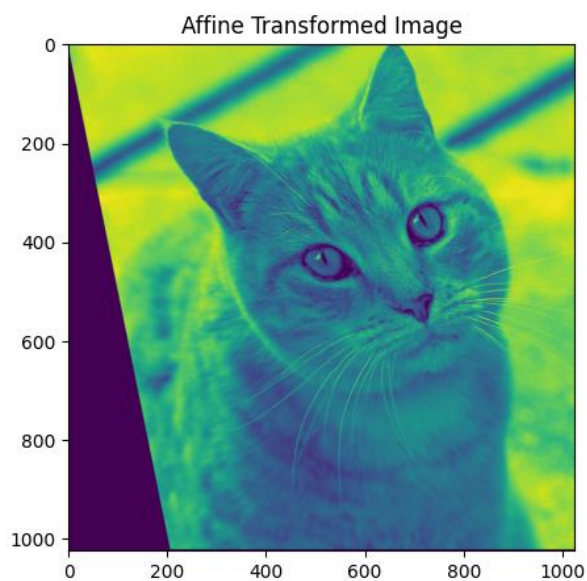
4. Rotation

```
1. rotated = sk.transform.rotate(image, 65)
2. plt.title("Rotated Image")
3. plt.imshow(rotated)
```



5. Affine

```
1. affined = sk.transform.warp(image,
sk.transform.AffineTransform(translation=(1,2), shear=0.2))
2.
3. plt.figure(figsize=(5, 5))
4. plt.title("Affine Transformed Image")
5. plt.imshow(affined)
6. plt.show()
```



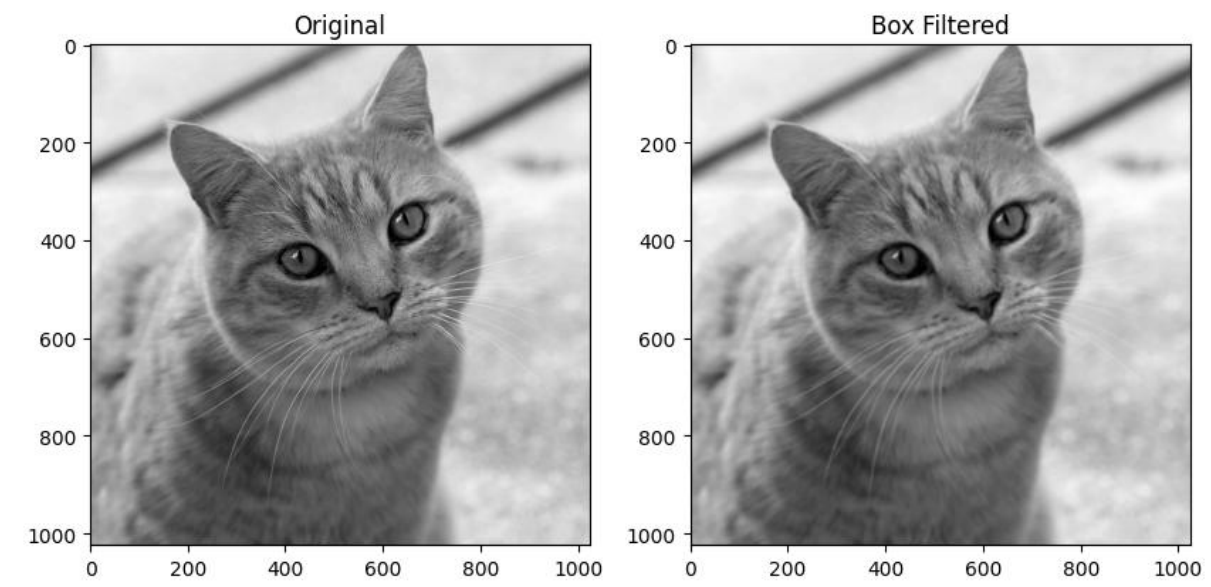
Exercise-5

Objective: Write a program to demonstrate Box Filter.

Code:

```
1. import cv2
2. import numpy as np
3. import matplotlib.pyplot as plt
4.
5. img = cv2.imread("../image.png", cv2.IMREAD_GRAYSCALE)
6.
7. # Apply box filter
8. box_filtered = cv2.boxFilter(img, ddepth=-1, ksize=(5, 5))
9.
10. # Display original and filtered images
11. plt.figure(figsize=(10, 5))
12. plt.subplot(1, 2, 1)
13. plt.title('Original')
14. plt.imshow(img, cmap='gray')
15.
16. plt.subplot(1, 2, 2)
17. plt.title('Box Filtered')
18. plt.imshow(box_filtered, cmap='gray')
19. plt.show()
```

Output:



Exercise-6

Objective: Write a program to create a shadow effect on an image.

Code:

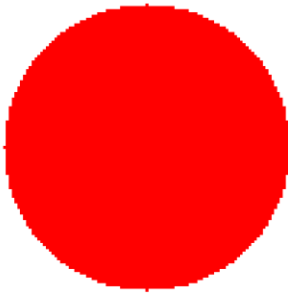
```
1. import cv2
2. import numpy as np
3. import matplotlib.pyplot as plt
4.
5. def shadow(image_path, shadow_offset=(10, 10), blur_ksize=(21, 21),
shadow_color=(0, 0, 0, 10)):
6.     # read the image with alpha channel
7.     image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
8.     plt.title("Original Image")
9.     plt.imshow(image)
10.    plt.show()
11.
12.    if image.shape[2] != 4:
13.        raise ValueError("Image must have an alpha channel")
14.
15.    h, w = image.shape[:2]
16.
17.    # Create shadow base from alpha channel
18.    alpha = image[:, :, 3]
19.    shadow = np.zeros((h, w, 4), dtype=np.uint8)
20.    shadow[:, :, :3] = shadow_color[:3]
21.    shadow[:, :, 3] = alpha
22.
23.    # Apply Gaussian blur to the shadow
24.    blurred_shadow = cv2.GaussianBlur(shadow, blur_ksize, 0)
25.
26.    # Create a larger canvas to accommodate the shadow offset
27.    canvas_h = h + shadow_offset[1]
28.    canvas_w = w + shadow_offset[0]
29.    canvas = np.zeros((canvas_h, canvas_w, 4), dtype=np.uint8)
30.
31.    # Print the blurred iamge
32.    canvas[shadow_offset[1]:shadow_offset[1]+h,
shadow_offset[0]:shadow_offset[0]+w] = blurred_shadow
33.
34.    # Overlay original image onto the canvas
35.    canvas[0:h, 0:w] = overlay_image_alpha(canvas[0:h, 0:w], image)
36.
37.    return canvas
38.
39. def overlay_image_alpha(background, foreground):
40.     alpha = foreground[:, :, 3] / 255.0
41.     for c in range(3):
42.         background[:, :, c] = (1 - alpha) * background[:, :, c] + alpha *
foreground[:, :, c]
43.     background[:, :, 3] = np.clip(foreground[:, :, 3] + background[:, :, 3] *
(1 - alpha), 0, 255)
```



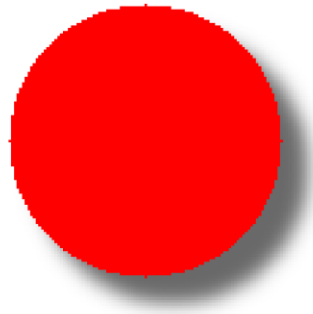
```
44.     return background.astype(np.uint8)
45.
46.
47. result = shadow("../image1.png", shadow_offset=(10, 10), blur_ksize=(21, 21),
shadow_color=(0, 0, 0, 10))
48. plt.title("Shadow Effect")
49. plt.imshow(result)
50. plt.show()
```

Output:

Original Image



Shadow Effect



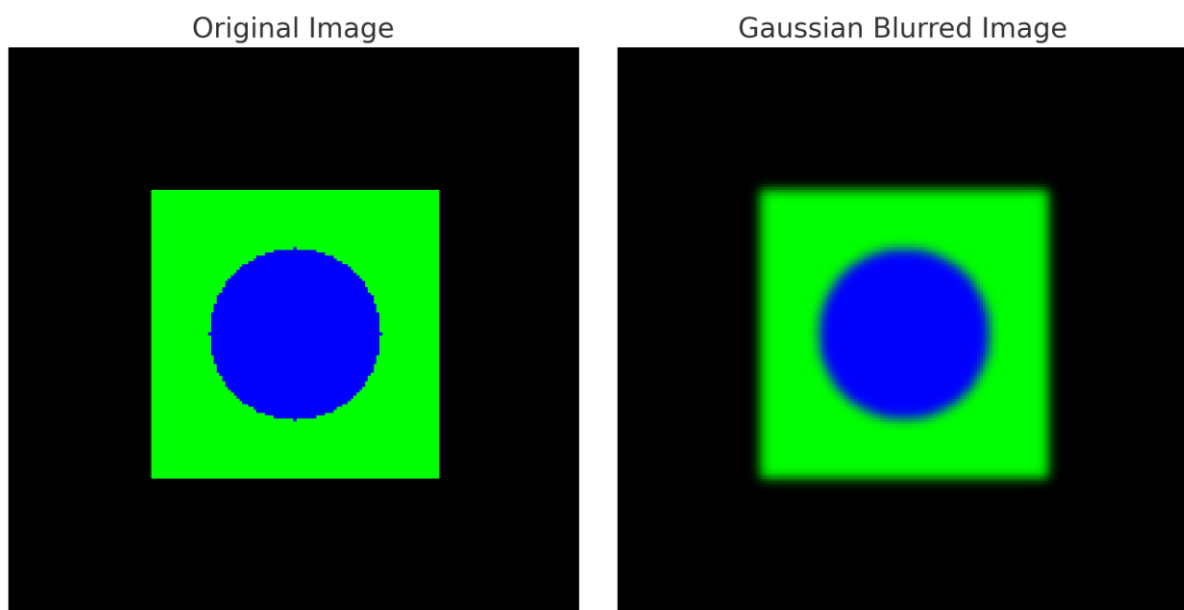
Exercise-7

Objective: Write a program to apply Gaussian Filtering to an image.

Code:

```
1. import cv2
2. import matplotlib.pyplot as plt
3.
4. img = '../image1.png'
5.
6. # Apply Gaussian filter
7. gaussian_filtered = cv2.GaussianBlur(img, ksize=(5, 5), sigmaX=0)
8.
9. # Display original and filtered images
10. plt.figure(figsize=(10, 5))
11. plt.subplot(1, 2, 1)
12. plt.title('Original')
13. plt.imshow(img, cmap='gray')
14.
15. plt.subplot(1, 2, 2)
16. plt.title('Gaussian Filtered')
17. plt.imshow(gaussian_filtered, cmap='gray')
18. plt.show()
```

Output:



Exercise-8

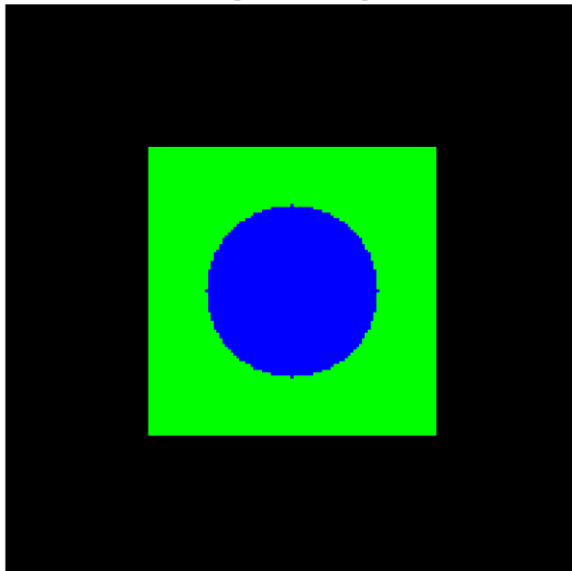
Objective: Write a program to apply Laplacian Filtering to an image.

Code:

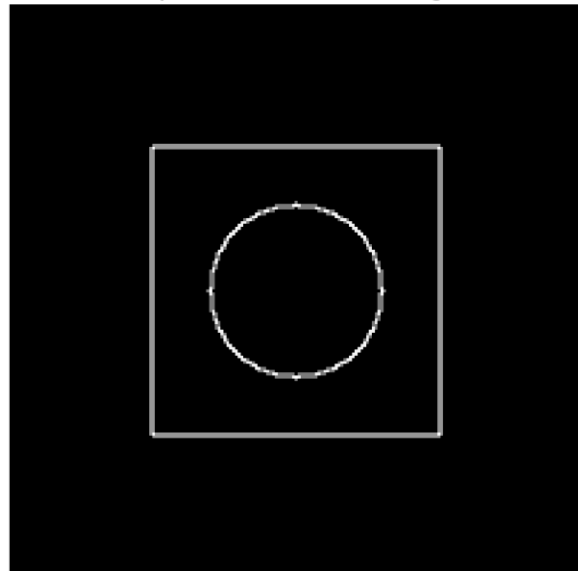
```
1. import cv2
2. import matplotlib.pyplot as plt
3.
4. img = '../image1.png'
5.
6. # Apply Laplacian filter
7. laplacian_filtered = cv2.Laplacian(img, ddepth=cv2.CV_64F)
8. # Display original and filtered images
9. plt.figure(figsize=(10, 5))
10. plt.subplot(1, 2, 1)
11. plt.title('Original')
12. plt.imshow(img, cmap='gray')
13. plt.subplot(1, 2, 2)
14. plt.title('Laplacian Filtered')
15. plt.imshow(laplacian_filtered, cmap='gray')
16. plt.show()
```

Output:

Original Image



Laplacian Filtered Image



Exercise-9

Objective: Write a program to create a Gaussian Pyramid for an image.

Code:

```
1. import cv2
2. import numpy as np
3. import matplotlib.pyplot as plt
4.
5. def create_gaussian_pyramid(image, levels=3):
6.     pyramid = [image]
7.     for i in range(levels - 1):
8.         # Apply Gaussian blur
9.         blurred = cv2.GaussianBlur(pyramid[i], (5, 5), 1)
10.        # Downsample the image
11.        downsampled = cv2.pyrDown(blurred)
12.        pyramid.append(downsampled)
13.
14.    return pyramid
15.
16. image = cv2.imread("../image.png")
17. image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
18.
19. # Create Gaussian Pyramid with 3 levels
20. pyramid = create_gaussian_pyramid(image_rgb, levels=3)
21.
22. # Plot the images in the pyramid
23. plt.figure(figsize=(15, 5))
24.
25. for i, img in enumerate(pyramid):
26.     plt.subplot(1, len(pyramid), i + 1)
27.     plt.title(f"Level {i + 1}")
28.     plt.imshow(img)
29. plt.tight_layout()
30. plt.show()
```

Output:



Exercise-10

Objective: Multiclass Image Classification using CNN.

Code:

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,  
# THEN FEEL FREE TO DELETE THIS CELL.
```

```
import kagglehub  
path = kagglehub.dataset_download('puneet6060/intel-image-classification')  
  
print('Data source import complete.')  
print("Path:", path)
```

1. Import the Required Libraries

```
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
%matplotlib inline  
  
import cv2  
import os  
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"  
import warnings  
warnings.filterwarnings('ignore')  
  
from sklearn.metrics import confusion_matrix, classification_report  
  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Activation, BatchNormalization, Conv2D  
, Dense, Dropout, Flatten, MaxPooling2D  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.losses import CategoricalCrossentropy  
from tensorflow.keras.regularizers import l2  
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

2. Load the Image Training and Validation Datasets

1. Get the Image Dataset Paths

```
train_dataset_path = path + '/seg_train/seg_train/'  
validation_dataset_path = path + '/seg_test/seg_test/'
```

2. Load Image Datasets and Apply Augmentations

Since the images present in the datasets are 150x150px in size, the image height and width are taken as 150, 150 respectively. The batch size value can be changed if required.

```

IMG_WIDTH = 150
IMG_HEIGHT = 150
BATCH_SIZE = 32

```

Loading the training dataset and applying augmentations on it.

```

train_datagen = ImageDataGenerator(rescale=1.0/255,
                                   zoom_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(train_dataset_path,
                                                    target_size=(IMG_WIDTH,
                                                                    IMG_HEIGHT),
                                                    batch_size=BATCH_SIZE,
                                                    class_mode='categorical',
                                                    shuffle=True)

```

Found 14034 images belonging to 6 classes.

Loading the validation dataset.

```

validation_datagen = ImageDataGenerator(rescale=1.0/255)
validation_generator = validation_datagen.flow_from_directory(validation_dataset_path,
                                                             target_size=(
IMG_WIDTH, IMG_HEIGHT),
                                                             batch_size=BATCH_SIZE,
                                                             class_mode='categorical',
                                                             shuffle=True)

```

Found 3000 images belonging to 6 classes.

iii. Get the Label Mappings

The labels dictionary is made in order to retrieve the class names against the label indices used for training the model

```

labels = {value: key for key, value in train_generator.class_indices.items()}

```

```

print("Label Mappings for classes present in the training and validation datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")

```

Label Mappings for classes present in the training and validation datasets

```

0 : buildings
1 : forest
2 : glacier
3 : mountain

```

```
4 : sea
5 : street
```

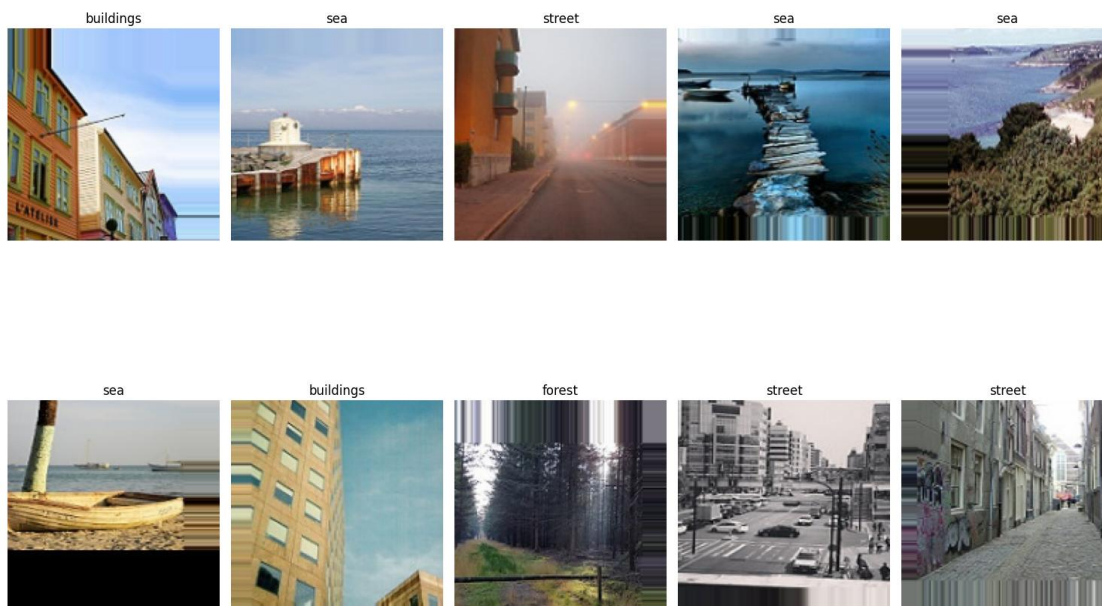
3. Plotting Sample Training Images

```
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(15, 12))
idx = 0

for i in range(2):
    for j in range(5):
        label = labels[np.argmax(train_generator[0][1][idx])]
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_generator[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1

plt.tight_layout()
plt.suptitle("Sample Training Images", fontsize=21)
plt.show()
```

Sample Training Images



4. Training a CNN Model

Since the training dataset is ready let's create a simple CNN Model to train on the image datasets

i. Create a CNN Model

```
def create_model():
    model = Sequential([
        Conv2D(filters=128, kernel_size=(5, 5), padding='valid', input_shape=(IMG_WIDTH, IMG_HEIGHT, 3)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
```

```

        BatchNormalization(),

        Conv2D(filters=64, kernel_size=(3, 3), padding='valid', kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=32, kernel_size=(3, 3), padding='valid', kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Flatten(),

        Dense(units=256, activation='relu'),
        Dropout(0.5),
        Dense(units=6, activation='softmax')
    ])

    return model

```

```
cnn_model = create_model()
```

```
print(cnn_model.summary())
```

```
Model: "sequential_1"
```

| Layer (type) | Output Shape | Param # |
|---|-----------------------|---------|
| conv2d_3 (Conv2D) | (None, 146, 146, 128) | 9,728 |
| activation_3 (Activation) | (None, 146, 146, 128) | 0 |
| max_pooling2d_3 (MaxPooling2D) | (None, 73, 73, 128) | 0 |
| batch_normalization_3 (BatchNormalization) | (None, 73, 73, 128) | 512 |
| conv2d_4 (Conv2D) | (None, 71, 71, 64) | 73,792 |

| | | |
|---|--------------------|-----------|
| activation_4 (Activation) | (None, 71, 71, 64) | 0 |
| max_pooling2d_4 (MaxPooling2D) | (None, 35, 35, 64) | 0 |
| batch_normalization_4 (BatchNormalization) | (None, 35, 35, 64) | 256 |
| conv2d_5 (Conv2D) | (None, 33, 33, 32) | 18,464 |
| activation_5 (Activation) | (None, 33, 33, 32) | 0 |
| max_pooling2d_5 (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| batch_normalization_5 (BatchNormalization) | (None, 16, 16, 32) | 128 |
| flatten_1 (Flatten) | (None, 8192) | 0 |
| dense_2 (Dense) | (None, 256) | 2,097,408 |
| dropout_1 (Dropout) | (None, 256) | 0 |
| dense_3 (Dense) | (None, 6) | 1,542 |

Total params: 2,201,830 (8.40 MB)

Trainable params: 2,201,382 (8.40 MB)

Non-trainable params: 448 (1.75 KB)

None

ii. Defining Callbacks

A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc)

a. Reduce Learning Rate on Plateau

Is used to reduce the learning rate when a metric has stopped improving.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=np.sqrt(0.1), patience=5, verbose=1)
```

iii. Defining the Optimizer

```
optimizer = Adam(learning_rate=0.001)
```

iv. Compile the Model

```
cnn_model.compile(optimizer=optimizer, loss=CategoricalCrossentropy(), metrics=['accuracy'])
```

v. Training the Model

```
history = cnn_model.fit(train_generator, epochs=50, validation_data=validation_generator, verbose=2, callbacks=[reduce_lr])
```

Epoch 1/50

439/439 - 65s - 149ms/step - accuracy: 0.4964 - loss: 2.1109 - val_accuracy: 0.5127 - val_loss: 1.1990 - learning_rate: 1.0000e-03

Epoch 2/50

439/439 - 48s - 110ms/step - accuracy: 0.6131 - loss: 1.0702 - val_accuracy: 0.5640 - val_loss: 1.3726 - learning_rate: 1.0000e-03

Epoch 3/50

439/439 - 48s - 110ms/step - accuracy: 0.6568 - loss: 0.9232 - val_accuracy: 0.6773 - val_loss: 0.9458 - learning_rate: 1.0000e-03

Epoch 4/50

439/439 - 49s - 111ms/step - accuracy: 0.6912 - loss: 0.8486 - val_accuracy: 0.7130 - val_loss: 0.8210 - learning_rate: 1.0000e-03

Epoch 5/50

439/439 - 48s - 109ms/step - accuracy: 0.7116 - loss: 0.8024 - val_accuracy: 0.7020 - val_loss: 0.7868 - learning_rate: 1.0000e-03

Epoch 6/50

439/439 - 48s - 109ms/step - accuracy: 0.7282 - loss: 0.7564 - val_accuracy: 0.7470 - val_loss: 0.7527 - learning_rate: 1.0000e-03

Epoch 7/50

439/439 - 48s - 110ms/step - accuracy: 0.7437 - loss: 0.7180 - val_accuracy: 0.8000 - val_loss: 0.5946 - learning_rate: 1.0000e-03

Epoch 8/50

439/439 - 48s - 110ms/step - accuracy: 0.7542 - loss: 0.6940 - val_accuracy: 0.7750 - val_loss: 0.6176 - learning_rate: 1.0000e-03

Epoch 9/50

439/439 - 48s - 110ms/step - accuracy: 0.7666 - loss: 0.6668 - val_accuracy: 0.3233 - val_loss: 6.4530 - learning_rate: 1.0000e-03

Epoch 10/50
439/439 - 48s - 110ms/step - accuracy: 0.7772 - loss: 0.6363 - val_accuracy: 0.8157 - val_loss: 0.5311 - learning_rate: 1.0000e-03

Epoch 11/50
439/439 - 48s - 110ms/step - accuracy: 0.7830 - loss: 0.6219 - val_accuracy: 0.7837 - val_loss: 0.6171 - learning_rate: 1.0000e-03

Epoch 12/50
439/439 - 48s - 109ms/step - accuracy: 0.7861 - loss: 0.6130 - val_accuracy: 0.7420 - val_loss: 0.7246 - learning_rate: 1.0000e-03

Epoch 13/50
439/439 - 48s - 109ms/step - accuracy: 0.7909 - loss: 0.5946 - val_accuracy: 0.6197 - val_loss: 1.1284 - learning_rate: 1.0000e-03

Epoch 14/50
439/439 - 48s - 110ms/step - accuracy: 0.7971 - loss: 0.5837 - val_accuracy: 0.8090 - val_loss: 0.5444 - learning_rate: 1.0000e-03

Epoch 15/50

Epoch 15: ReduceLROnPlateau reducing learning rate to 0.00031622778103685084.
439/439 - 48s - 110ms/step - accuracy: 0.8096 - loss: 0.5662 - val_accuracy: 0.7547 - val_loss: 0.7665 - learning_rate: 1.0000e-03

Epoch 16/50
439/439 - 49s - 112ms/step - accuracy: 0.8318 - loss: 0.4980 - val_accuracy: 0.8703 - val_loss: 0.3983 - learning_rate: 3.1623e-04

Epoch 17/50
439/439 - 48s - 109ms/step - accuracy: 0.8394 - loss: 0.4691 - val_accuracy: 0.8630 - val_loss: 0.4069 - learning_rate: 3.1623e-04

Epoch 18/50
439/439 - 48s - 110ms/step - accuracy: 0.8464 - loss: 0.4613 - val_accuracy: 0.8727 - val_loss: 0.3951 - learning_rate: 3.1623e-04

Epoch 19/50
439/439 - 48s - 110ms/step - accuracy: 0.8440 - loss: 0.4555 - val_accuracy: 0.8333 - val_loss: 0.5072 - learning_rate: 3.1623e-04

Epoch 20/50
439/439 - 48s - 110ms/step - accuracy: 0.8454 - loss: 0.4509 - val_accuracy: 0.8790 - val_loss: 0.3827 - learning_rate: 3.1623e-04

Epoch 21/50
439/439 - 48s - 109ms/step - accuracy: 0.8499 - loss: 0.4347 - val_accuracy: 0.8083 - val_loss: 0.6546 - learning_rate: 3.1623e-04

Epoch 22/50
439/439 - 48s - 110ms/step - accuracy: 0.8572 - loss: 0.4179 - val_accuracy: 0.8757 - val_loss: 0.4188 - learning_rate: 3.1623e-04

Epoch 23/50
439/439 - 48s - 110ms/step - accuracy: 0.8546 - loss: 0.4227 - val_accuracy: 0.8743 - val_loss: 0.4059 - learning_rate: 3.1623e-04

Epoch 24/50
439/439 - 49s - 111ms/step - accuracy: 0.8556 - loss: 0.4202 - val_accuracy: 0.8563 - val_loss: 0.4641 - learning_rate: 3.1623e-04

Epoch 25/50
439/439 - 48s - 110ms/step - accuracy: 0.8575 - loss: 0.4151 - val_accuracy: 0.8737 - val_loss: 0.3679 - learning_rate: 3.1623e-04

Epoch 26/50
439/439 - 48s - 110ms/step - accuracy: 0.8588 - loss: 0.4144 - val_accuracy: 0.8707 - val_loss: 0.3899 - learning_rate: 3.1623e-04

Epoch 27/50
439/439 - 48s - 109ms/step - accuracy: 0.8598 - loss: 0.4095 - val_accuracy: 0.8860 - val_loss: 0.3508 - learning_rate: 3.1623e-04

Epoch 28/50
439/439 - 49s - 111ms/step - accuracy: 0.8655 - loss: 0.4053 - val_accuracy: 0.8560 - val_loss: 0.4347 - learning_rate: 3.1623e-04

Epoch 29/50
439/439 - 48s - 109ms/step - accuracy: 0.8625 - loss: 0.3991 - val_accuracy: 0.8303 - val_loss: 0.5277 - learning_rate: 3.1623e-04

Epoch 30/50
439/439 - 48s - 109ms/step - accuracy: 0.8647 - loss: 0.3964 - val_accuracy: 0.8847 - val_loss: 0.3669 - learning_rate: 3.1623e-04

Epoch 31/50
439/439 - 48s - 108ms/step - accuracy: 0.8645 - loss: 0.3888 - val_accuracy: 0.8660 - val_loss: 0.4414 - learning_rate: 3.1623e-04

Epoch 32/50

Epoch 32: ReduceLROnPlateau reducing learning rate to 0.00010000000639606199.
439/439 - 47s - 108ms/step - accuracy: 0.8720 - loss: 0.3785 - val_accuracy: 0.8730 - val_loss: 0.4176 - learning_rate: 3.1623e-04

Epoch 33/50
439/439 - 48s - 109ms/step - accuracy: 0.8777 - loss: 0.3583 - val_accuracy: 0.8903 - val_loss: 0.3283 - learning_rate: 1.0000e-04

Epoch 34/50
439/439 - 48s - 109ms/step - accuracy: 0.8813 - loss: 0.3551 - val_accuracy: 0.8913 - val_loss: 0.3452 - learning_rate: 1.0000e-04

Epoch 35/50
439/439 - 47s - 108ms/step - accuracy: 0.8802 - loss: 0.3503 - val_accuracy: 0.8847 - val_loss: 0.3761 - learning_rate: 1.0000e-04

Epoch 36/50
439/439 - 47s - 107ms/step - accuracy: 0.8833 - loss: 0.3446 - val_accuracy: 0.8840 - val_loss: 0.3650 - learning_rate: 1.0000e-04

Epoch 37/50
439/439 - 47s - 107ms/step - accuracy: 0.8816 - loss: 0.3385 - val_accuracy: 0.8917 - val_loss: 0.3435 - learning_rate: 1.0000e-04

Epoch 38/50

Epoch 38: ReduceLROnPlateau reducing learning rate to 3.1622778103685084e-05.
439/439 - 48s - 108ms/step - accuracy: 0.8842 - loss: 0.3398 - val_accuracy: 0.8650 - val_loss: 0.4605 - learning_rate: 1.0000e-04

Epoch 39/50
439/439 - 47s - 108ms/step - accuracy: 0.8845 - loss: 0.3349 - val_accuracy: 0.8853 - val_loss: 0.3488 - learning_rate: 3.1623e-05

Epoch 40/50
439/439 - 47s - 108ms/step - accuracy: 0.8854 - loss: 0.3262 - val_accuracy: 0.8820 - val_loss: 0.3739 - learning_rate: 3.1623e-05

Epoch 41/50
439/439 - 47s - 108ms/step - accuracy: 0.8901 - loss: 0.3225 - val_accuracy: 0.8920 - val_loss: 0.3374 - learning_rate: 3.1623e-05

Epoch 42/50
439/439 - 48s - 108ms/step - accuracy: 0.8901 - loss: 0.3231 - val_accuracy: 0.8940 - val_loss: 0.3280 - learning_rate: 3.1623e-05

Epoch 43/50
 439/439 - 47s - 107ms/step - accuracy: 0.8890 - loss: 0.3229 - val_accuracy: 0.8933 - val_loss: 0.3381 - learning_rate: 3.1623e-05
 Epoch 44/50
 439/439 - 48s - 109ms/step - accuracy: 0.8868 - loss: 0.3286 - val_accuracy: 0.8920 - val_loss: 0.3412 - learning_rate: 3.1623e-05
 Epoch 45/50
 439/439 - 48s - 108ms/step - accuracy: 0.8896 - loss: 0.3338 - val_accuracy: 0.8943 - val_loss: 0.3311 - learning_rate: 3.1623e-05
 Epoch 46/50
 439/439 - 48s - 109ms/step - accuracy: 0.8891 - loss: 0.3221 - val_accuracy: 0.8930 - val_loss: 0.3368 - learning_rate: 3.1623e-05
 Epoch 47/50

Epoch 47: ReduceLROnPlateau reducing learning rate to 1.0000000409520217e-05.

439/439 - 48s - 109ms/step - accuracy: 0.8886 - loss: 0.3198 - val_accuracy: 0.8903 - val_loss: 0.3477 - learning_rate: 3.1623e-05

Epoch 48/50
 439/439 - 48s - 110ms/step - accuracy: 0.8898 - loss: 0.3246 - val_accuracy: 0.8947 - val_loss: 0.3303 - learning_rate: 1.0000e-05

Epoch 49/50
 439/439 - 47s - 108ms/step - accuracy: 0.8882 - loss: 0.3241 - val_accuracy: 0.8953 - val_loss: 0.3321 - learning_rate: 1.0000e-05

Epoch 50/50
 439/439 - 48s - 108ms/step - accuracy: 0.8893 - loss: 0.3253 - val_accuracy: 0.8933 - val_loss: 0.3319 - learning_rate: 1.0000e-05

5. Plotting the Model Metrics

i. Plotting training and validation accuracy, loss and learning rate

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

train_loss = history.history['loss']
val_loss = history.history['val_loss']

learning_rate = history.history['learning_rate']

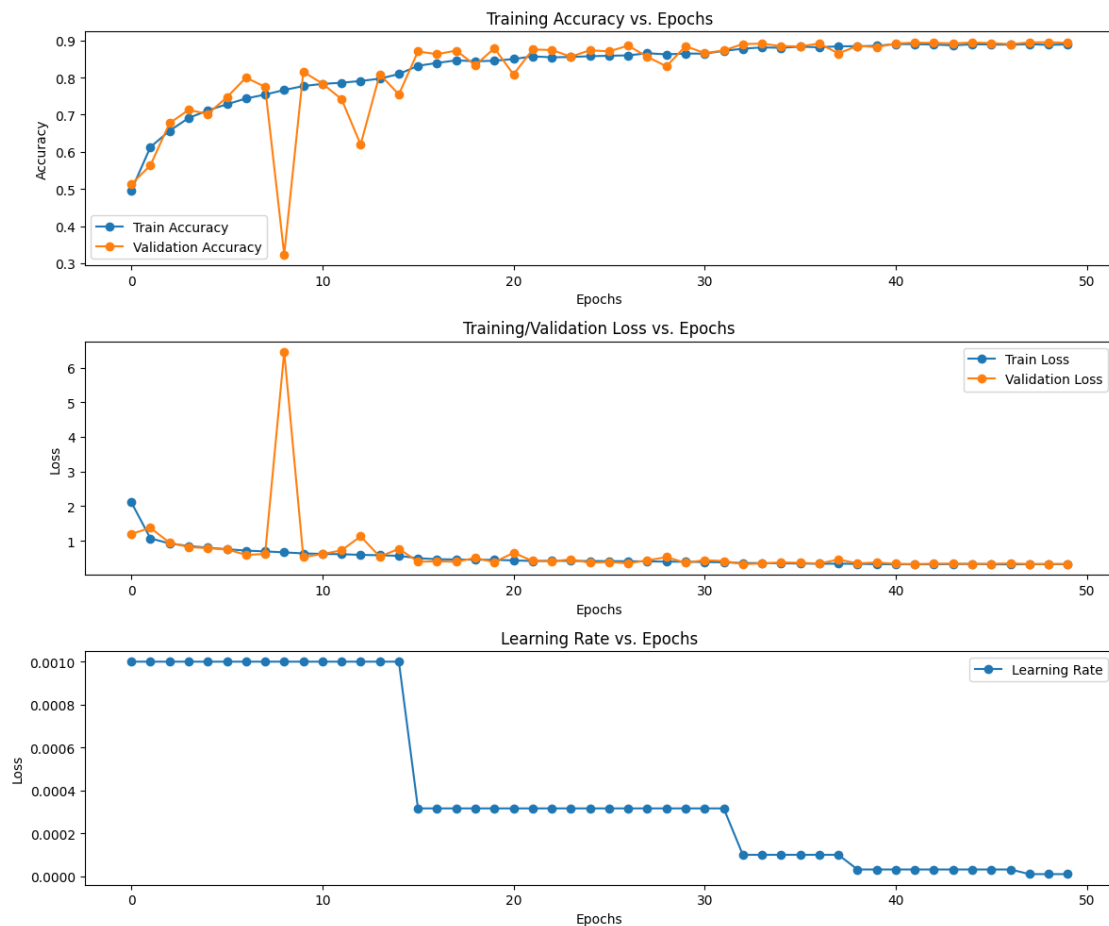
fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))

ax[0].set_title('Training Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')

ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')
```

```
ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')
```

```
plt.tight_layout()
plt.show()
```



6. Testing the Model on Test Set

Testing the model on the validation dataset because a separate dataset for testing is not available.

```
test_dataset = path + '/seg_test/seg_test/'
```

```
test_datagen = ImageDataGenerator(rescale=1.0/255)
```

```
test_generator = test_datagen.flow_from_directory(test_dataset,
                                                  shuffle=False,
                                                  batch_size=BATCH_SIZE,
                                                  target_size = (IMG_WIDTH,
IMG_HEIGHT),
                                                  class_mode='categorical')
```

Found 3000 images belonging to 6 classes.

7. Model Prediction on the Test Dataset

```
predictions = cnn_model.predict(test_generator)
```

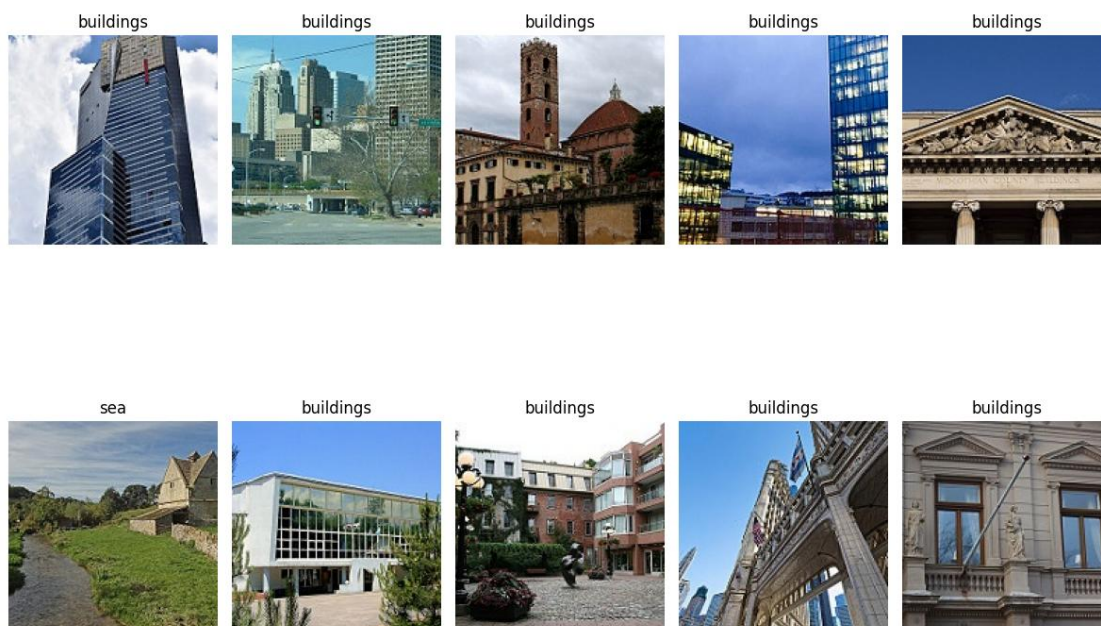
```
94/94 ————— 4s 30ms/step
```

```
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(12, 10))  
idx = 0
```

```
for i in range(2):  
    for j in range(5):  
        predicted_label = labels[np.argmax(predictions[idx])]  
        ax[i, j].set_title(f"{predicted_label}")  
        ax[i, j].imshow(test_generator[0][0][idx])  
        ax[i, j].axis("off")  
        idx += 1
```

```
plt.tight_layout()  
plt.suptitle("Test Dataset Predictions", fontsize=20)  
plt.show()
```

Test Dataset Predictions



```
test_loss, test_accuracy = cnn_model.evaluate(test_generator, batch_size=BATCH_SIZE)
```

```
94/94 ————— 2s 19ms/step - accuracy: 0.9104 - loss: 0.2821
```

```
print(f"Test Loss:      {test_loss}")  
print(f"Test Accuracy: {test_accuracy}")
```

```
Test Loss:      0.3318774402141571  
Test Accuracy: 0.8933333158493042
```

The test loss and test accuracy is the same as validation loss and validation accuracy at the last step since the testing and validation datasets are same.

8. Plotting the Classification Metrics

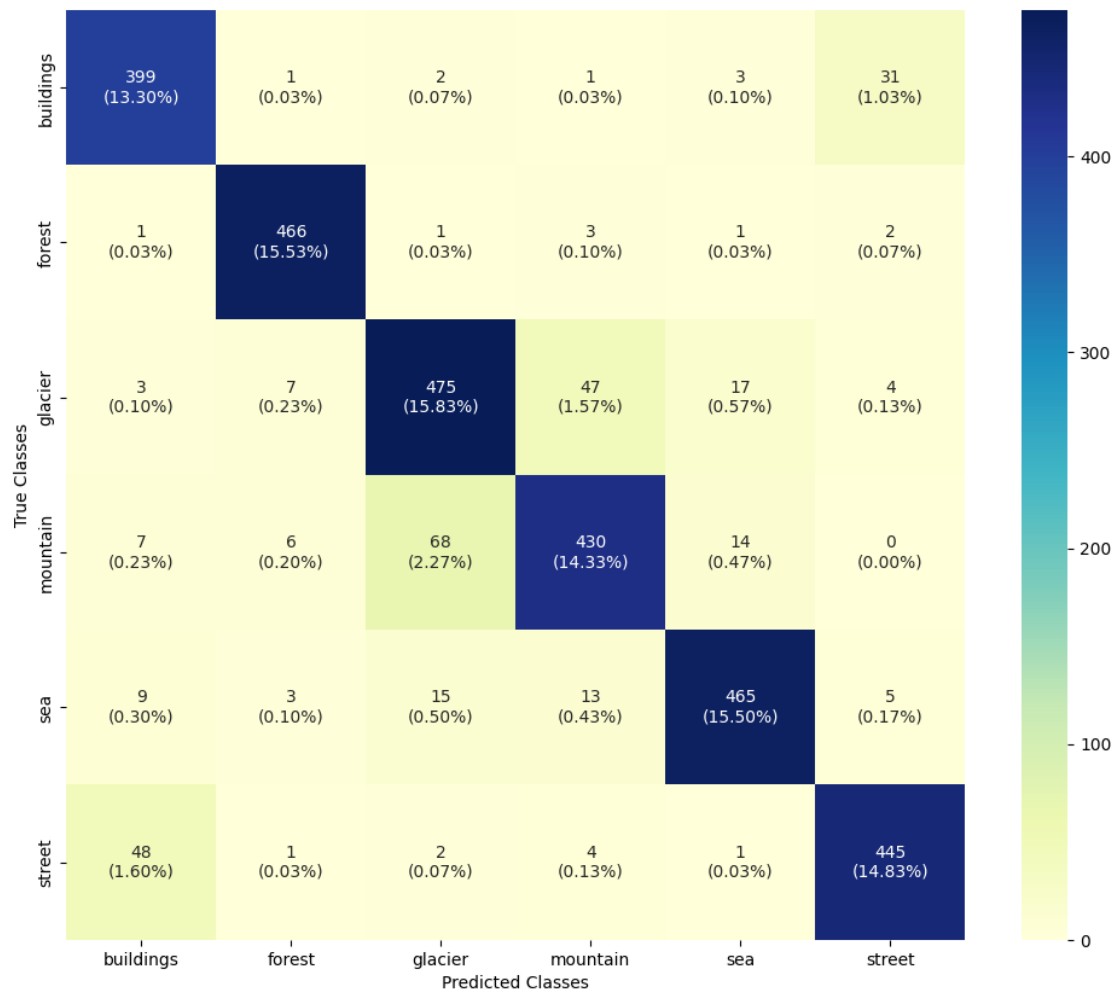
i. Confusion Matrix

```
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes

cf_mtx = confusion_matrix(y_true, y_pred)

group_counts = ["{0:0.0f}".format(value) for value in cf_mtx.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_mtx.flatten()
                    / np.sum(cf_mtx)]
box_labels = [f"{v1}\n({v2})" for v1, v2 in zip(group_counts, group_percentages)]
box_labels = np.asarray(box_labels).reshape(6, 6)

plt.figure(figsize = (12, 10))
sns.heatmap(cf_mtx, xticklabels=labels.values(), yticklabels=labels.values(),
            cmap="YlGnBu", fmt="", annot=box_labels)
plt.xlabel('Predicted Classes')
plt.ylabel('True Classes')
plt.show()
```



```
print(classification_report(y_true, y_pred, target_names=labels.values()))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| buildings | 0.85 | 0.91 | 0.88 | 437 |
| forest | 0.96 | 0.98 | 0.97 | 474 |
| glacier | 0.84 | 0.86 | 0.85 | 553 |
| mountain | 0.86 | 0.82 | 0.84 | 525 |
| sea | 0.93 | 0.91 | 0.92 | 510 |
| street | 0.91 | 0.89 | 0.90 | 501 |
| accuracy | | | 0.89 | 3000 |
| macro avg | 0.89 | 0.90 | 0.89 | 3000 |
| weighted avg | 0.89 | 0.89 | 0.89 | 3000 |

9. Wrong Predictions

Let's see where the model has given wrong predictions and what were the actual predictions on those images.

```
errors = (y_true - y_pred != 0)
y_true_errors = y_true[errors]
y_pred_errors = y_pred[errors]
```



```

test_images = test_generator.fileNames
test_img = np.asarray(test_images)[errors]

fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(12, 10))
idx = 0

for i in range(2):
    for j in range(5):
        idx = np.random.randint(0, len(test_img))
        true_index = y_true_errors[idx]
        true_label = labels[true_index]
        predicted_index = y_pred_errors[idx]
        predicted_label = labels[predicted_index]
        ax[i, j].set_title(f"True Label: {true_label} \n Predicted Label: {predicted_label}")
        img_path = os.path.join(test_dataset, test_img[idx])
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax[i, j].imshow(img)
        ax[i, j].axis("off")

plt.tight_layout()
plt.suptitle('Wrong Predictions made on test set', fontsize=20)
plt.show()

```

Wrong Predictions made on test set

