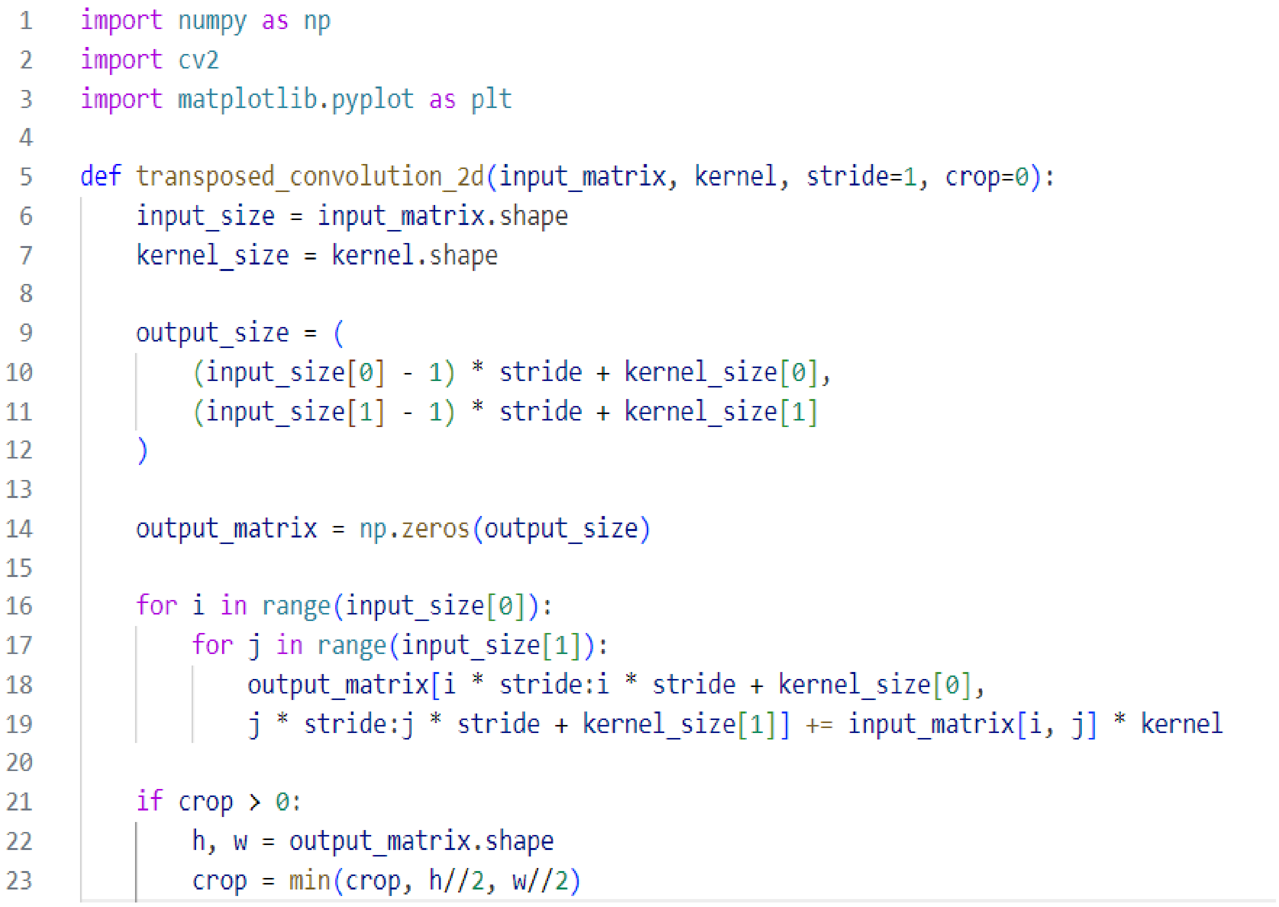
AYUSH KIRAN BADGUJAR (230259)

EE655: Computer Vision and Deep Learning

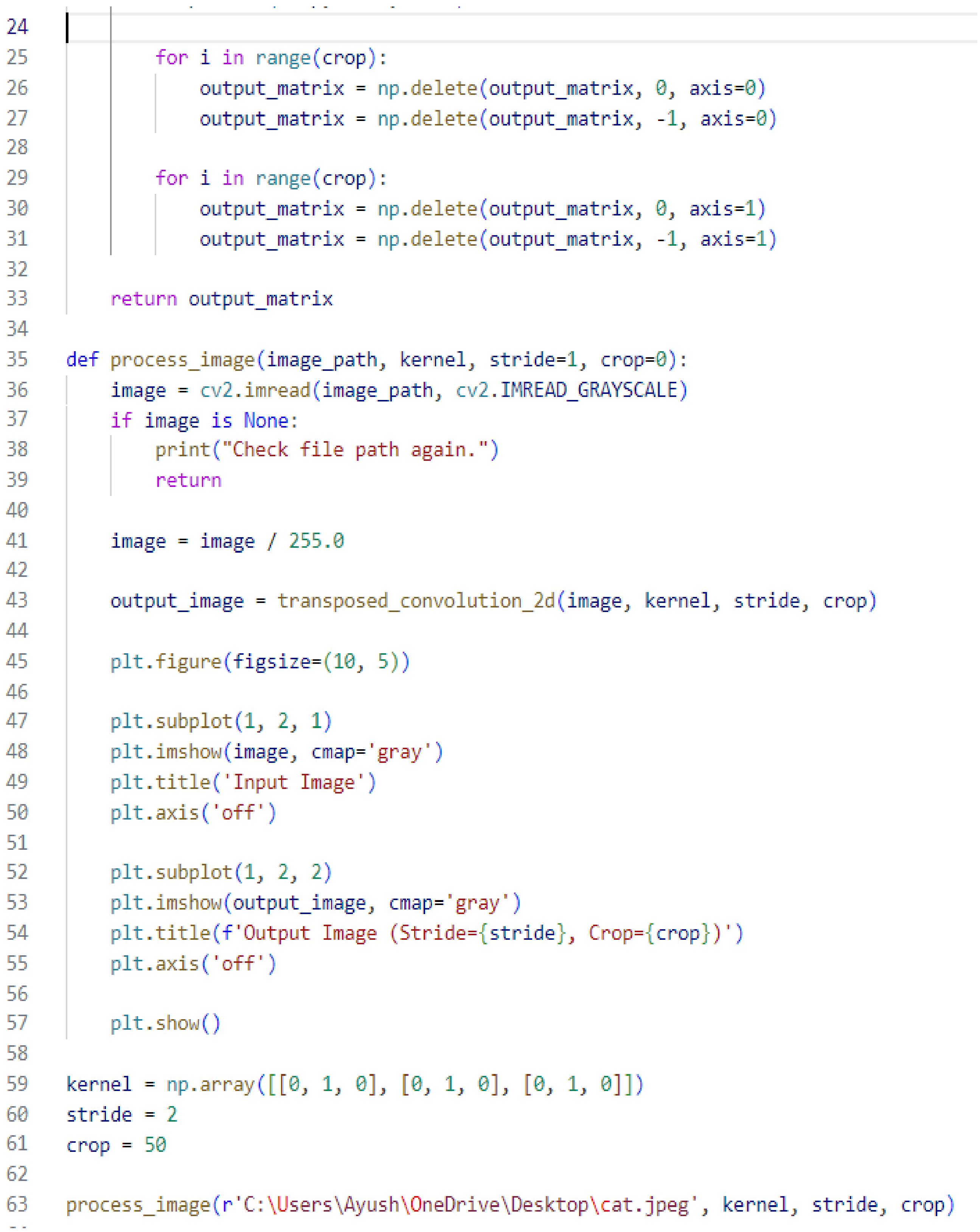
Homework 4

Q1) Implement transposed convolution from scratch as a generic function that takes the following inputs:

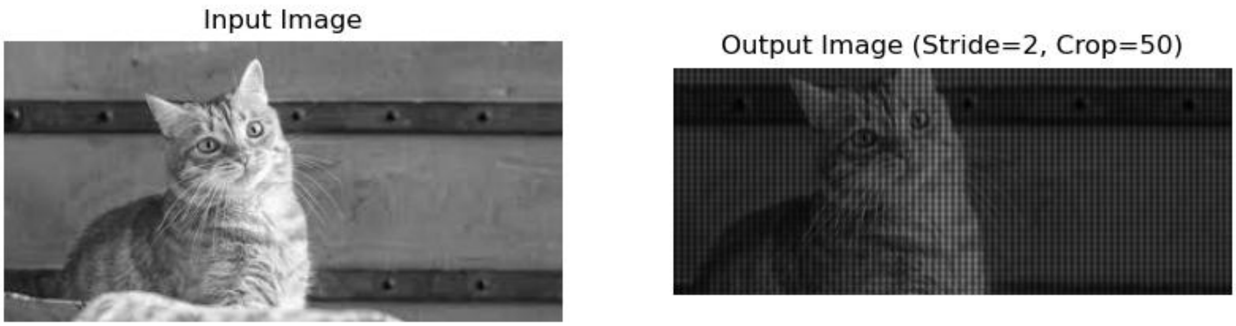
1. A 2D matrix on which transposed convolution needs to be performed.
2. A 2D kernel used for transposed convolution.
3. A stride parameter.
4. A crop parameter.



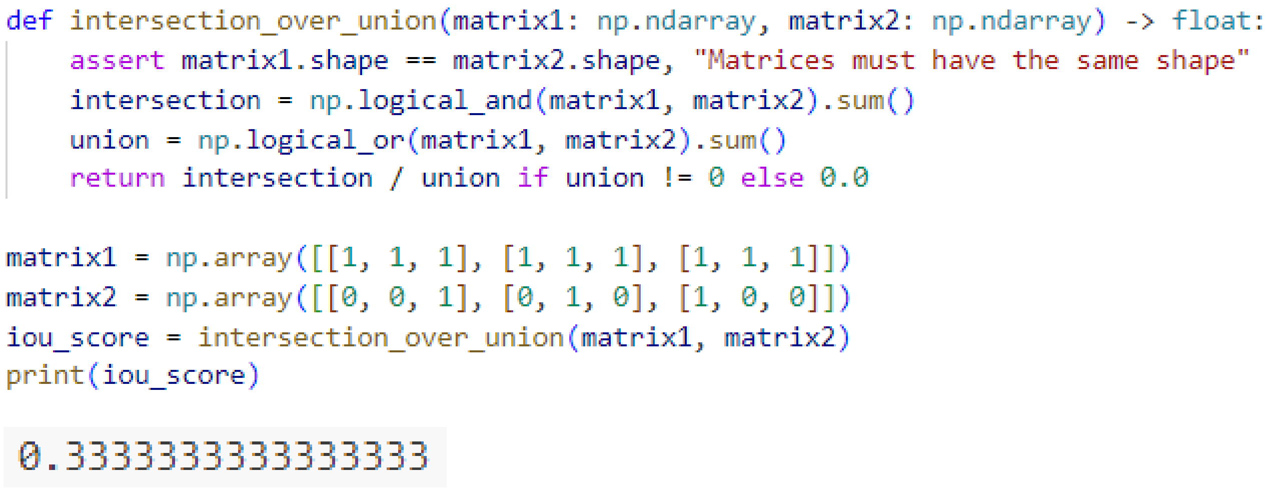
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def transposed_convolution_2d(input_matrix, kernel, stride=1, crop=0):
    input_size = input_matrix.shape
    kernel_size = kernel.shape

    output_size = (
        (input_size[0] - 1) * stride + kernel_size[0],
        (input_size[1] - 1) * stride + kernel_size[1]
    )

    output_matrix = np.zeros(output_size)

    for i in range(input_size[0]):
        for j in range(input_size[1]):
            output_matrix[i * stride:i * stride + kernel_size[0],
            j * stride:j * stride + kernel_size[1]] += input_matrix[i, j] * kernel

    if crop > 0:
        h, w = output_matrix.shape
        crop = min(crop, h//2, w//2)
```



```python

        for i in range(crop):
            output_matrix = np.delete(output_matrix, 0, axis=0)
            output_matrix = np.delete(output_matrix, -1, axis=0)

        for i in range(crop):
            output_matrix = np.delete(output_matrix, 0, axis=1)
            output_matrix = np.delete(output_matrix, -1, axis=1)

    return output_matrix

def process_image(image_path, kernel, stride=1, crop=0):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print("Check file path again.")
        return

    image = image / 255.0

    output_image = transposed_convolution_2d(image, kernel, stride, crop)

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Input Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(output_image, cmap='gray')
    plt.title(f'Output Image (Stride={stride}, Crop={crop})')
    plt.axis('off')

    plt.show()

kernel = np.array([[0, 1, 0], [0, 1, 0], [0, 1, 0]])
stride = 2
crop = 50

process_image(r'C:\Users\Ayush\OneDrive\Desktop\cat.jpeg', kernel, stride, crop)
```



Q2) Implement the Intersection over Union (IoU) metric from scratch as a generic function that takes two binary matrices as inputs.



```
def intersection_over_union(matrix1: np.ndarray, matrix2: np.ndarray) -> float:
    assert matrix1.shape == matrix2.shape, "Matrices must have the same shape"
    intersection = np.logical_and(matrix1, matrix2).sum()
    union = np.logical_or(matrix1, matrix2).sum()
    return intersection / union if union != 0 else 0.0

matrix1 = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
matrix2 = np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0]])
iou_score = intersection_over_union(matrix1, matrix2)
print(iou_score)
```

```
0.3333333333333333
```