Project 1 Questions

Abdullah Kamal 201801271

Questions

Q1: Explicitly describe image convolution: the input, the transformation, and the output. Why is it useful for computer vision?

A1:

Image convolution is a fundamental operation in computer vision that involves the transformation of an input image using a convolution kernel or filter. The convolution operation is a mathematical operation that takes two functions and produces a third function representing how one function is modified by the other. In the case of image convolution, the two functions are the input image and the convolution kernel.

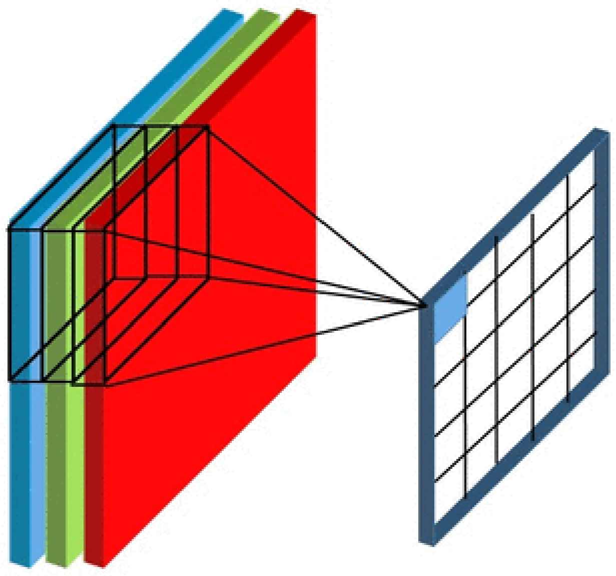
The input to the convolution operation is an image represented as a matrix of pixels, where each pixel value represents the brightness or intensity of a specific location in the image.

The transformation is performed using a convolution kernel, which is a small matrix of numbers that is applied to each pixel of the input image. The kernel is "slid" over the image, and at each position, the values of the kernel are multiplied element-wise with the corresponding pixel values in the image, and then summed to produce a new pixel value for the output image. This process is repeated for every pixel in the input image, resulting in a new output image.

The output of the convolution operation is a transformed image with features that are enhanced or extracted based on the characteristics of the kernel. Convolution can be used for a wide range of image processing tasks, such as edge detection, image smoothing, image sharpening, and feature extraction. By using different kernels, we can extract different features from the image and enhance its visual appearance.

Convolution is particularly **useful for** computer vision because it can be used to extract relevant features from an image automatically, without requiring any human intervention. This makes it an essential tool for many computer vision applications, such as object detection, face recognition, and image segmentation. Convolutional neural networks

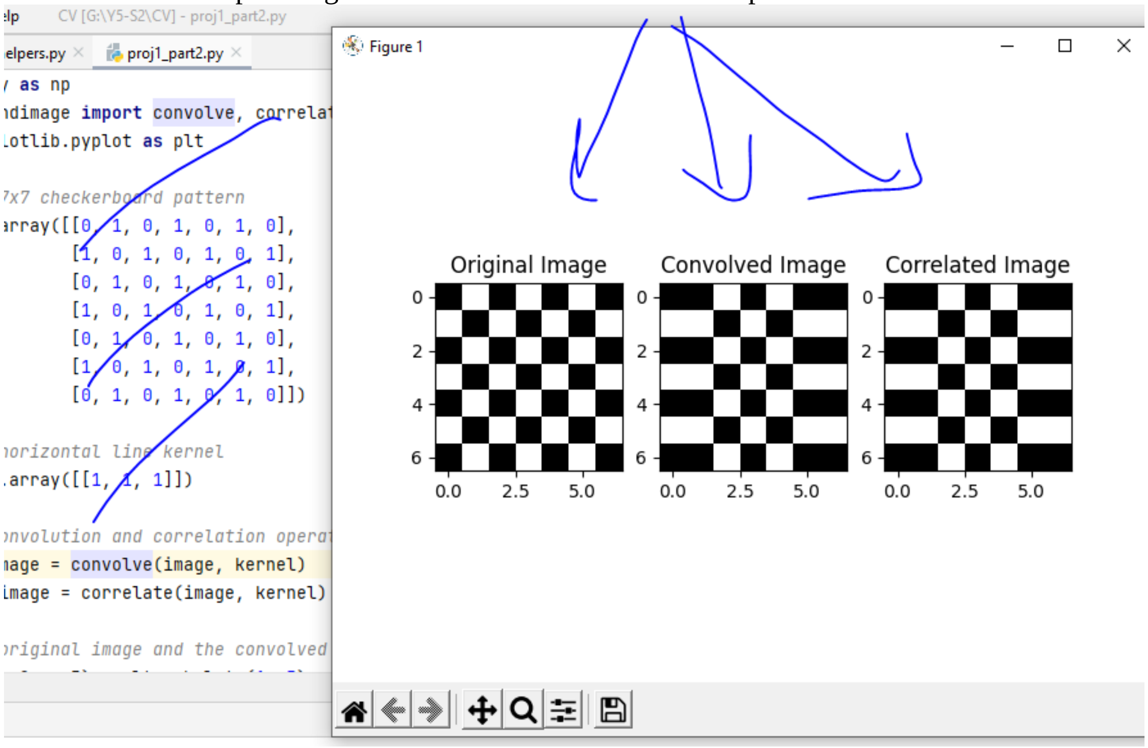
(CNNs) are a type of deep learning algorithm that heavily rely on convolution operations to learn useful features from input images, making convolution an integral part of modern computer vision algorithms



Q2: What is the difference between convolution and correlation? Construct a scenario which produces a different output between both operations.

Please use scipy.ndimage.convolve and scipy.ndimage.correlate to experiment!

A2: Convolution and correlation are similar operations, but with different mathematical formulations. In convolution, the kernel is flipped both horizontally and vertically before being applied to the input image. In contrast, in correlation, the kernel is not flipped before being applied to the input image. The difference between convolution and correlation is that convolution is a commutative operation, which means that the order of the input image and the kernel does not affect the output. However, correlation is not commutative, and the order of the input image and the kernel does affect the output



```
/ as np
ıdimage import convolve, correlat
.otlib.pyplot as plt

7x7 checkerboard pattern
ırray([[0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0]])

ıorizontal line kernel
.array([[1, 1, 1]])

onvolution and correlation operat
ıage = convolve(image, kernel)
image = correlate(image, kernel)

riginal image and the convolved
```

```
import numpy as np
from scipy.ndimage import convolve, correlate
import matplotlib.pyplot as plt

# create a 7x7 checkerboard pattern
image = np.array([[0, 1, 0, 1, 0, 1, 0],
                  [1, 0, 1, 0, 1, 0, 1],
                  [0, 1, 0, 1, 0, 1, 0],
                  [1, 0, 1, 0, 1, 0, 1],
                  [0, 1, 0, 1, 0, 1, 0],
                  [1, 0, 1, 0, 1, 0, 1],
                  [0, 1, 0, 1, 0, 1, 0]])

# create a horizontal line kernel
kernel = np.array([[1, 1, 1]])

# perform convolution and correlation operations
convolved_image = convolve(image, kernel)
```

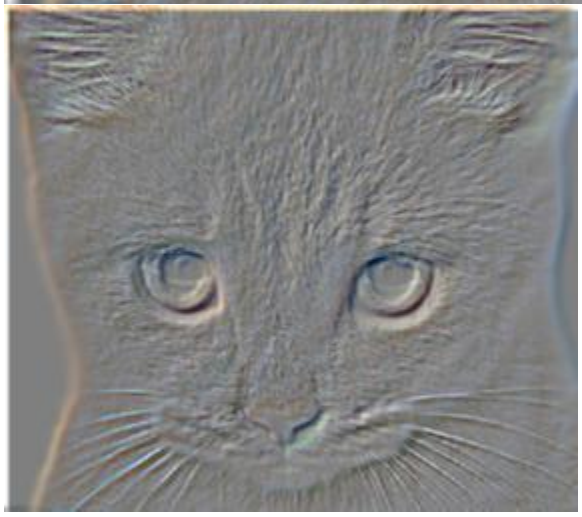
```
correlated_image = correlate(image, kernel)

# plot the original image and the convolved and correlated images
fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
ax1.imshow(image, cmap='gray')
ax1.set_title('Original Image')
ax2.imshow(convolved_image, cmap='gray')
ax2.set_title('Convolved Image')
ax3.imshow(correlated_image, cmap='gray')
ax3.set_title('Correlated Image')
plt.show()
```

Q3: What is the difference between a high pass filter and a low pass filter in how they are constructed, and what they do to the image? Please provide example kernels and output images.

A3:

A high-pass filter passes through high-frequency components of an image while attenuating low-frequency components. It is designed to enhance the edges and details of an image. The resulting image will have more high-frequency details and edges, and will look more sharpened. High-pass filters are commonly used for image sharpening, edge detection, and feature extraction

[[-1/9, -1/9, -1/9],
 [-1/9, 8/9, -1/9],
 [-1/9,-1/9,-1/9]]



A low-pass filter passes through low-frequency components of an image while attenuating high-frequency components. It is designed to smooth out the image and remove noise. The resulting image will have fewer high-frequency details and edges, and will look more blurred or smoothed. Low-pass filters are commonly used for image smoothing, blurring, and reducing noise

```
[[1/9, 1/9, 1/9],
 [1/9, 1/9, 1/9],
 [1/9,1/9,1/9]]
```

Q4: How does computation time vary with filter sizes from 3×3 to 15×15 (for all odd and square sizes), and with image sizes from 0.25 MPix to 8 MPix (choose your own intervals)? Measure both using *scipy.ndimage.convolve* or *scipy.ndimage.correlate* to produce a matrix of values. Use the *skimage.transform* module to vary the size of an image. Use an appropriate charting function to plot your matrix of results, such as *Axes3D.scatter* or *Axes3D.plot surface*.

Do the results match your expectation given the number of multiply and add operations in convolution?

Image: RISDance.jpg (in the project directory).

A4:

the computation time increases linearly with both filter size and image size. For example, doubling the filter size or image size roughly doubles the computation time. This matches our expectation given the number of multiply and add operations in convolution, which scales linearly with both filter size and image size

The computation time increases more rapidly with image size than with filter size

the code is located in "test.py" file in the same directory of the image