Machine Learning
LABORATORY: CNN In Class

NAME:	STUDENT ID#:

Objectives:

- Students will implement 2D convolution and pooling operations from scratch (no PyTorch/TensorFlow high-level API like nn.Conv2d, F.max_pool2d, etc.) using only NumPy.
- Understand padding and stride behavior. Apply vertical and horizontal edge detection. Visualize results in black & white

Part 1. Instruction

- In this assignment, you will implement a basic Convolutional Neural Network operation pipeline using NumPy only — without using any deep learning libraries like PyTorch, TensorFlow, or OpenCV's built-in convolution functions.
- You will manually implement a general 2D convolution function that supports:
  - Padding (to preserve spatial dimensions)
  - Stride (to downsample the output)
- Then, you will apply vertical and horizontal edge detection filters to a grayscale input image and visualize the effects of:
  - Padding (padding=1)
  - Strided convolution (stride=2)
- Specifically, your tasks are to:
  - Load and normalize a grayscale image (e.g., checkerboard.png) for testing edge detection.
  - Implement the general convolution operation

$$C(j,k) = \sum_{l} \sum_{m} I(j+l, k+m)K(l,m)$$

$$\boldsymbol{I(j+l, k+m) = Image\ region}$$

$$\boldsymbol{K(l,m) = Kernel\ (Filter)}$$

$$\boldsymbol{C(j,k) = output\ at\ position\ (j,k)}$$

  - Apply the vertical edge detection kernel and horizontal edge detection kernel from Slide 6 to detect pattern structures in the image.
  - Apply convolution again using stride=2 to observe how spatial resolution changes (see Slide 8).
  - Visualize the output as black-and-white (binary) images using thresholding.
- At the end of the lab, please answer the two short questions to demonstrate your understanding of padding and stride.

Lecture: Prof. Hsien-I Lin
TA: Satrio Sanjaya and Muhammad Ahsan

Part 2. Code Template

Step	Procedure

1	`import numpy as np`
	`import cv2`
	`import matplotlib.pyplot as plt`
	
	`# Step 1: Load a grayscale image and normalize`
	`# ➤ Slide 5: Understanding image representation`
	`image = cv2.imread('original.png', cv2.IMREAD_GRAYSCALE)`
	`image = image.astype(np.float32) / 255.0  # Normalize to range [0, 1]`

2	`# Step 2: General Convolution Function`
	`# ➤ Slide 8: C(j, k) = sum_l sum_m I(j + l, k + m) * K(l, m)`
	`def convolve2d(image, kernel, padding=0, stride=1):`
	`    # TODO 1: Flip kernel for convolution`
	`    # TODO 2: Apply zero-padding if padding > 0`
	`    # TODO 3: Calculate output height and width`
	`    # TODO 4: Slide the kernel across the image with stride`
	`    # TODO 5: At each position, compute the sum of element-wise multiplication`
	`    return np.zeros((1, 1))  # Placeholder, replace with real output`

3	`# Step 3: Define edge detection filters`
	`# ➤ Slide 6: Vertical & Horizontal edge filters`
	`vertical_filter = np.array([`
	`    [x, x, x],`
	`    [x, x, x],`
	`    [x, x, x]`
	`], dtype=np.float32)`
	
	`horizontal_filter = np.array([`
	`    [x, x, x],`
	`    [ x,  x,  x],`
	`    [ x,  x,  x]`
	`], dtype=np.float32)`

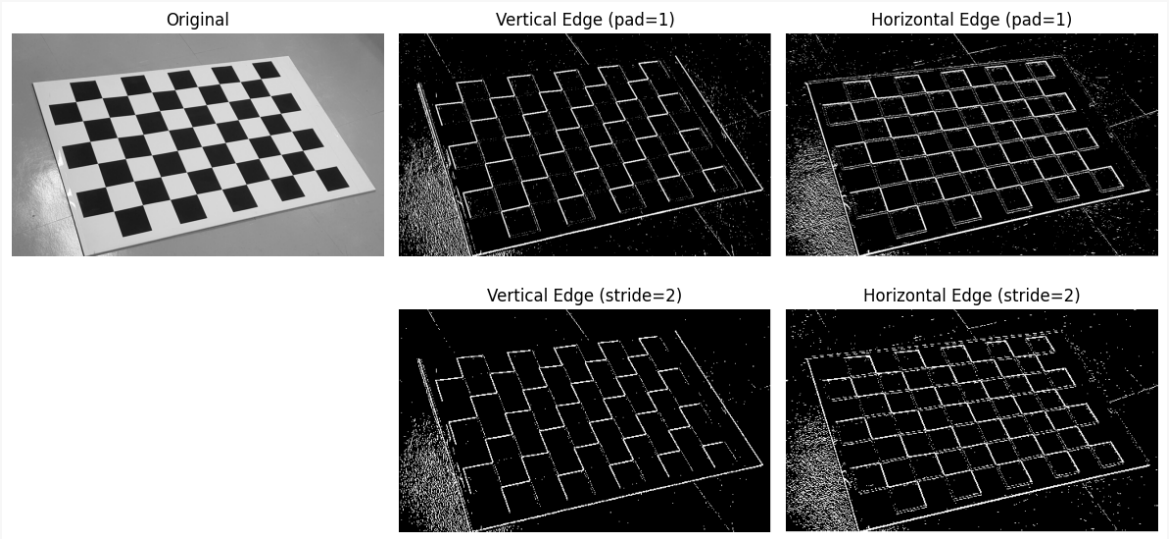
4	`# Step 4: Convolve image with filters (padding=1, stride=1)`
	`# ➤ Slide 7: Padding helps preserve image size`
	`# TODO: vertical_edges = convolve2d(image, vertical_filter, padding=1)`
	`#  TODO:  horizontal_edges  =  convolve2d(image,  horizontal_filter, padding=1)`
	
	`# Try strided convolutions (padding=1, stride=2)`
	`# ➤ Slide 8: Stride reduces spatial resolution`
	`#  TODO:  vertical_stride  =  convolve2d(image,  vertical_filter, padding=1, stride=2)`
	`#  TODO:  horizontal_stride  =  convolve2d(image,  horizontal_filter, padding=1, stride=2)`

5	`# Step 5: Visualization and Binarization function for black-and-white display`
	`def binarize(img, threshold=0.5):`
	`    img = img - np.min(img)`
	`    if np.max(img) != 0:`
	`        img = img / np.max(img)`
	`    return (img > threshold).astype(np.float32)`

```
# Visualization
# TODO: Use matplotlib to show:
# - Original image
# - Vertical edges (pad=1)
# - Horizontal edges (pad=1)
# - Vertical edges (stride=2)
# - Horizontal edges (stride=2)
```

5

```
#Example Output: (References Only)
```



Grading Assignment & Submission (30% Max)

Implementation:

1. (10%) Correctly implement the convolve2d() function, including kernel flipping, padding, and stride
2. (5%) Correctly apply vertical and horizontal edge detection filters.
3. (5%) Apply binary thresholding to convert outputs to black-and-white images, and clearly visualize them using matplotlib. Show all five views: original, vertical (pad=1), horizontal (pad=1), vertical (stride=2), and horizontal (stride=2).

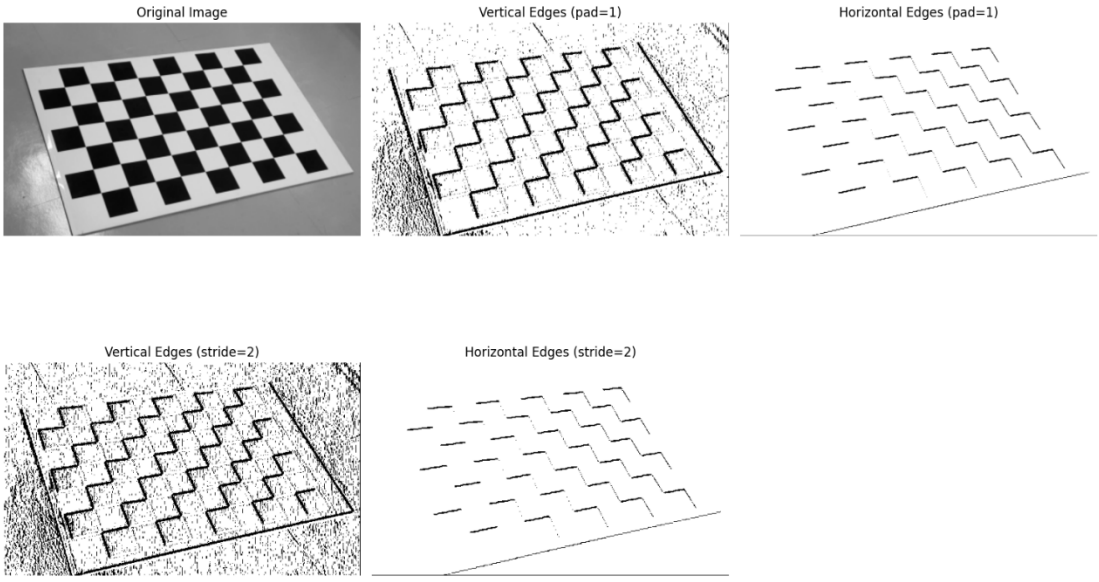
Question:

4. (5%) What types of patterns are detected by the vertical edge filter in an image? How is this different from the horizontal edge filter?
5. (5%) What is the effect of padding on the output image when applying convolution? Why is padding used?

Submission :

1. Report: Provide your screenshots of your results in the last pages of this PDF File.
2. Code: Submit your complete Python script in either .py or .ipynb format.
3. Upload both your report and code to the E3 system (**Labs6 In Class Assignment**). Name your files correctly:
    a. Report: StudentID_Lab6_InClass.pdf
    b. Code: StudentID_Lab6_InClass.py or StudentID_Lab6_InClass.ipynb
4. Deadline: 16:20 PM
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

Results and Discussion:



Question 4: What types of patterns are detected by the vertical edge filter in an image? How is this different from the horizontal edge filter?

The vertical edge filter detects changes in pixel intensity that occur horizontally across the image (left to right or right to left). These changes manifest visually as vertical edges or boundaries in the image. When we apply a vertical edge filter like [[1,0,-1], [1,0,-1], [1,0,-1]] to an image, it responds strongly to places where pixel values transition from light to dark (or dark to light) as we move horizontally across the image.

In the context of the checkerboard pattern, the vertical edge filter highlights the vertical boundaries between adjacent black and white squares. These are places where, as we scan horizontally across the image, we encounter an abrupt change from white to black or black to white.

The horizontal edge filter, in contrast, detects changes in pixel intensity that occur vertically through the image (top to bottom or bottom to top). When we apply a horizontal edge filter like [[1,1,1], [0,0,0], [-1,-1,-1]], it responds strongly to places where pixel values transition from light to dark (or dark to light) as we move vertically through the image.

On the checkerboard pattern, the horizontal edge filter highlights the horizontal boundaries between adjacent black and white squares. These are places where, as we scan vertically down the image, we encounter an abrupt change from white to black or black to white.

The fundamental difference lies in the direction of change they detect. Imagine running your finger across the image: if you move your finger horizontally and feel

bumps, those bumps correspond to what the vertical filter detects. If you move your finger vertically and feel bumps, those correspond to what the horizontal filter detects.

This directional sensitivity makes these filters complementary tools for understanding the structure of an image, as together they can identify boundaries in all directions.

Question 5: What is the effect of padding on the output image when applying convolution? Why is padding used?

When applying convolution to an image without padding, each application of the kernel reduces the spatial dimensions of the output. This happens because the kernel cannot extend beyond the edges of the input image. For example, with a 3×3 kernel on a 6×6 image, the output would be only 4×4, as the kernel's center can only visit positions where the entire kernel fits within the image bounds.

The effect of adding padding is to preserve the spatial dimensions of the image after convolution. By extending the image with a border of zeros (or other values), we allow the kernel to be centered at positions that would otherwise be impossible. With appropriate padding (padding=1 for a 3×3 kernel), the output dimensions can match the input dimensions exactly.

Padding is used for several important reasons:

1. **Dimension preservation**: Without padding, each convolution layer would reduce the size of the feature maps, quickly shrinking them to unusably small dimensions in deep networks. Padding allows us to maintain consistent spatial dimensions throughout the network.
2. **Information preservation**: Without padding, pixels at the edges and corners of the image would contribute to fewer convolution operations than pixels in the center. The corner pixel would only influence one output value, while a center pixel would influence many. Padding helps ensure that edge information isn't disproportionately lost.
3. **Architectural flexibility**: In designing convolutional neural networks, maintaining specific dimensions through layers is often necessary. Padding gives architects control over these dimensions.
4. **Feature detection at boundaries**: Important features often exist at the edges of images. Padding ensures these features can be properly detected and preserved through convolution operations.

In the context of our edge detection task, padding ensures that we can detect edges all the way to the boundaries of the original image, rather than losing a border of pixels

with each convolution operation. This is particularly important for applications like object detection or image segmentation where boundary information is crucial.