Laboratory work 2

(Variant 3)

Geometrical Transformations, Convolution, and Maxpooling

Objective: the purpose of this laboratory work is to study basic geometric transformations and core CNN operations.

The submission includes: ipynb and the report file. The report should include step-by-step description, results, and screenshots. Format the report according to the template.

Task 1. Geometrical transformations affine
Resize the image to 256×256. Use source points (x, y): (30, 30), (200, 40), (40, 210) and destination points (x, y): (40, 50), (210, 60), (60, 220). Compute the affine matrix using cv2.getAffineTransform and apply it using cv2.warpAffine. Use the same order for source and destination points: point1 to point1, point2 to point2, point3 to point3. Display side by side the original image and the affine transformed image.

Task 2. Affine matrix printing and point verification
Print the affine matrix M (2×3) from Task 1. Use three points (x, y) on the original image: (30, 30), (128, 128), (220, 180). Transform these points using cv2.transform() with matrix M and print the coordinates before and after. Draw the original points on the original image and the transformed points on the affine image.

Task 3. Nonlinear geometric transformation using remap
Create a nonlinear warp using cv2.remap. Build map_x and map_y as float32 arrays of size 256×256. Apply a sinusoidal horizontal shift using amplitude A = 12 pixels and frequency f = 2 periods across the image width. Use interpolation cv2.INTER_LINEAR and borderMode cv2.BORDER_REFLECT. Apply cv2.remap and display side by side the original image and the nonlinear transformed image.

Task 4. Pixel value check for nonlinear transform
Use pixel coordinate (x, y) = (70, 160) where x is the column index and y is the row index. Print the RGB value at (70, 160) in the original image and print the RGB value at (70, 160) in the nonlinear transformed image from Task 3. Add 1–2 sentences explaining why pixel values at the same coordinate may differ after nonlinear transformation.

Task 5. Convolution with OpenCV blur
Create a 3×3 blur kernel and apply convolution using cv2.filter2D. Convert the original image and the blurred image to grayscale. Display the grayscale original image and the grayscale blurred image side by side. Print the 5×5 grayscale patch from rows 90 to 94 and columns 90 to 94 for the original grayscale image and for the blurred grayscale image.

Task 6. Convolution with OpenCV sharpening
Create a 3×3 sharpening kernel and apply convolution using cv2.filter2D. Convert the sharpened result to grayscale. Display the grayscale original image and the grayscale sharpened image side by side. Print the same 5×5 grayscale patch from rows 90 to 94 and columns 90 to 94 for the sharpened grayscale image. Write 2–3 sentences comparing blur and sharpen.

Task 7. Convolution from scratch NumPy
Implement 2D convolution manually using NumPy for the blur kernel from Task 5. Use zero padding so that the output size matches 256×256. Display your manual convolution result and the cv2.filter2D blur result side by side. Print the same 5×5 grayscale patch from rows 90 to 94 and columns 90 to 94 from both results.

Task 8. Maxpooling NumPy
Using the blurred grayscale image from Task 5, implement maxpooling with window size 2×2 and stride 2. Print the input shape and output shape. Print the 4×4 grayscale block from rows 90 to 93 and columns 90 to 93 from the input and print the corresponding 2×2 pooled block from rows 45 to 46 and columns 45 to 46 from the output because stride = 2. Display the input grayscale image and the pooled image side by side.

Task 9. Manual calculation of convolution
Compute the convolution output value manually for the given input patch and kernel. Show all 9 multiplications and the final sum. Do not use OpenCV or NumPy convolution functions.
Input patch
2 1 3
0 4 2
1 2 0
Kernel
1 1 1
1 1 1
1 1 1

Task 10. Manual calculation of maxpooling
Compute maxpooling manually for the given 4×4 matrix using window size 2×2 and stride 2. Show the four 2×2 windows and the chosen maximum for each window. Write the final 2×2 output matrix. Do not use OpenCV or NumPy pooling functions.
Input matrix
3 1 4 2
0 5 2 1
6 2 1 7
4 3 0 5