Kazakh-British Technical University

Introduction to Computer Vision

Laboratory Work #2

Geometrical Transformations, Convolution, and Maxpooling (Variant 3)

Full Name: Sekenova Balym

ID: 23B031433

Almaty

February 9, 2026

Contents

1 Introduction

The purpose of this laboratory work is to study basic geometric transformations and core operations used in convolutional neural networks (CNNs). The work focuses on understanding how image data is transformed at both the geometric and pixel levels using classical computer vision techniques.

During this laboratory work, affine and nonlinear geometric transformations were applied to an input image. In addition, fundamental CNN-related operations, including convolution, sharpening, and maxpooling, were implemented and analyzed. Both built-in OpenCV functions and manual NumPy implementations were used in order to better understand the internal mechanics of these operations.

Special attention was given to numerical verification of results through pixel value analysis and local image patches. Visual and numerical comparisons were used to confirm the correctness of each operation. This laboratory work provides a practical foundation for understanding image preprocessing and feature extraction techniques commonly used in computer vision and deep learning applications.

2 Task 1: Geometrical transformations affine

In this task, an affine geometric transformation was applied to the input image. First, the image was loaded using OpenCV and resized to a fixed resolution of 256×256 pixels, as required by the assignment.

Three source points were selected on the resized image with coordinates (30, 30), (200, 40), and (40, 210). For each source point, a corresponding destination point was defined as (40, 50), (210, 60), and (60, 220), respectively. The order of points was strictly preserved, such that the first source point was mapped to the first destination point, the second to the second, and the third to the third.

Using these point correspondences, the affine transformation matrix was computed with the function `cv2.getAffineTransform`. The resulting matrix has dimensions 2×3 and represents a combination of geometric operations including translation, scaling, rotation, and shear.

The computed affine matrix was then applied to the image using `cv2.warpAffine`. The output image retained the same spatial resolution of 256×256 pixels. Finally, the original image and the affine-transformed image were displayed side by side in order to visually analyze the effect of the affine transformation.

Figure 1: Original image and affine transformed image. Task 1

3 Task 2: Affine matrix printing and point verification

In this task, the affine transformation matrix M obtained in Task 1 was printed to explicitly examine its numerical values. The matrix has dimensions 2×3 and defines the geometric relationship between the original image and the affine-transformed image.

To verify the correctness of the affine transformation, three points were selected on the original image with coordinates (30, 30), (128, 128), and (220, 180). These points were transformed using the function `cv2.transform()` together with the affine matrix M. The coordinates of the points were printed before and after the transformation to provide numerical confirmation of the mapping.

For visual verification, the selected points were drawn on the original image, and their transformed counterparts were drawn on the affine-transformed image. This visual comparison confirms that the affine matrix correctly maps points from the original image to their corresponding locations in the transformed image.

Table 1: Affine transformation matrix M obtained in Task 2

m_{11}	m_{12}	t_x
0.996721311	0.0557377049	8.42622951
m_{21}	m_{22}	t_y
0.00327868852	0.944262295	21.5737705

Table 2: Original points and their transformed coordinates using affine matrix M. Task 2

Point #	**Original** (x, y)	**Transformed** (x', y')
1	(30, 30)	(40.0, 50.0)
2	(128, 128)	(143.14098, 142.85901)
3	(220, 180)	(237.7377, 192.2623)

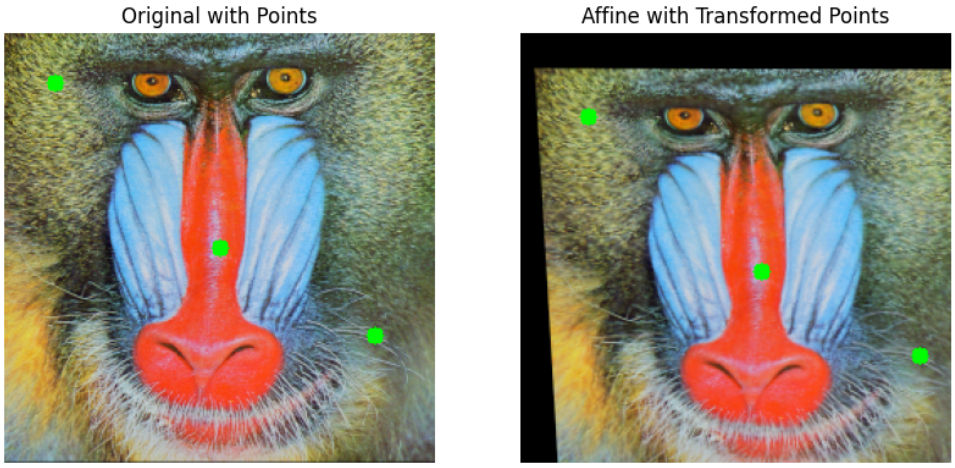

Figure 2: Original image with selected points and affine-transformed image with corresponding transformed points. Task 2

4 Task 3: Nonlinear geometric transformation using remap

In this task, a nonlinear geometric transformation was implemented using the OpenCV function `cv2.remap`. Unlike affine transformations, which are defined by a single transformation matrix, remapping allows pixel-wise control of image coordinates.

Two mapping arrays, `map_x` and `map_y`, were created as `float32` matrices of size 256×256. These arrays define, for each pixel in the output image, the corresponding source coordinates in the original image. Initially, `map_x` and `map_y` were set to represent the identity mapping.

A nonlinear horizontal displacement was then applied by modifying `map_x` using a sinusoidal function. The amplitude of the displacement was set to $A = 12$ pixels, and the frequency was set to $f = 2$, resulting in two full sinusoidal periods across the image width. The vertical coordinates defined by `map_y` were left unchanged.

The nonlinear transformation was applied using `cv2.remap` with linear interpolation (`cv2.INTER_LINEAR`) and reflected border handling (`cv2.BORDER_REFLECT`). The original image and the nonlinearly transformed image were displayed side by side to visually

analyze the effect of the nonlinear warping.

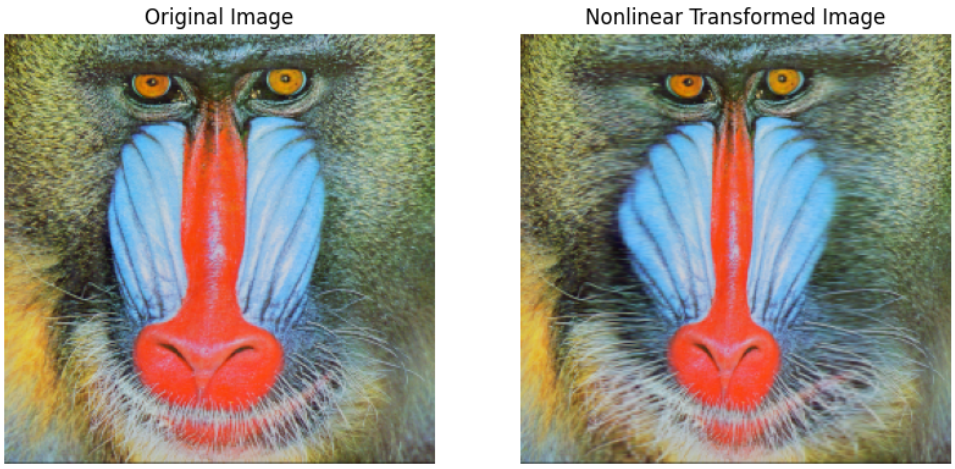

Figure 3: Original image and nonlinear transformed image obtained using sinusoidal remapping. Task 3

5 Task 4: Pixel value check for nonlinear transform

In this task, the RGB values of a single pixel were examined before and after the nonlinear geometric transformation. The pixel with coordinates $(x, y) = (70, 160)$, where x represents the column index and y the row index, was selected for analysis.

The RGB value at this coordinate was printed for the original image and for the nonlinearly transformed image obtained in Task 3. Although the spatial coordinates are the same, the pixel values differ.

This difference occurs because the nonlinear remapping changes the source location from which each output pixel is sampled. As a result, the pixel at $(70, 160)$ in the transformed image corresponds to a different location in the original image and its value is computed using interpolation.

Table 3: RGB values at pixel coordinate $(x, y) = (70, 160)$ before and after nonlinear transformation. Task 4

Image	**R**	**G**	**B**
Original image	69	71	44
Nonlinear transformed image	73	70	49

6 Task 5: Convolution with OpenCV blur

In this task, image smoothing was performed using convolution with a blur kernel. A 3×3 averaging kernel was created, where all kernel values are equal and the sum of the elements equals one. This type of kernel performs local averaging of pixel intensities.

The convolution was applied to the original image using the OpenCV function `cv2.filter2D`. As a result, each output pixel was computed as the average of its neighboring pixels, leading to a smoothing effect.

Both the original image and the blurred image were then converted to grayscale to simplify numerical analysis. The grayscale original image and the grayscale blurred image were displayed side by side for visual comparison.

To quantitatively analyze the effect of the blur operation, a 5×5 grayscale patch was extracted from rows 90 to 94 and columns 90 to 94 of both images. The pixel values before and after blurring were printed and compared.

Table 4: 5×5 grayscale patch from the original image (rows 90–94, columns 90–94). Task 5

181	189	187	187	182
183	188	187	188	182
176	189	191	195	188
180	181	188	196	190
187	182	188	192	195

Table 5: 5×5 grayscale patch from the blurred image (rows 90–94, columns 90–94). Task 5

181	185	187	185	184
181	185	189	187	186
181	185	189	189	188
182	185	189	191	190
183	184	187	191	191

Tables 4 and 5 show the numerical effect of the blur convolution, where local intensity variations are reduced due to averaging of neighboring pixels.

7 Task 6: Convolution with OpenCV sharpening

In this task, image sharpening was performed using convolution with a sharpening kernel. A 3×3 sharpening kernel was defined and applied to the original image using the OpenCV function `cv2.filter2D`. This type of kernel enhances edges by emphasizing intensity differences between a pixel and its neighbors.

After applying the sharpening filter, the resulting image was converted to grayscale. The grayscale original image and the grayscale sharpened image were displayed side by side to visually analyze the effect of sharpening.

To quantitatively evaluate the sharpening operation, a 5×5 grayscale patch from rows 90 to 94 and columns 90 to 94 was extracted from the sharpened image. The pixel values of this patch were printed and analyzed.

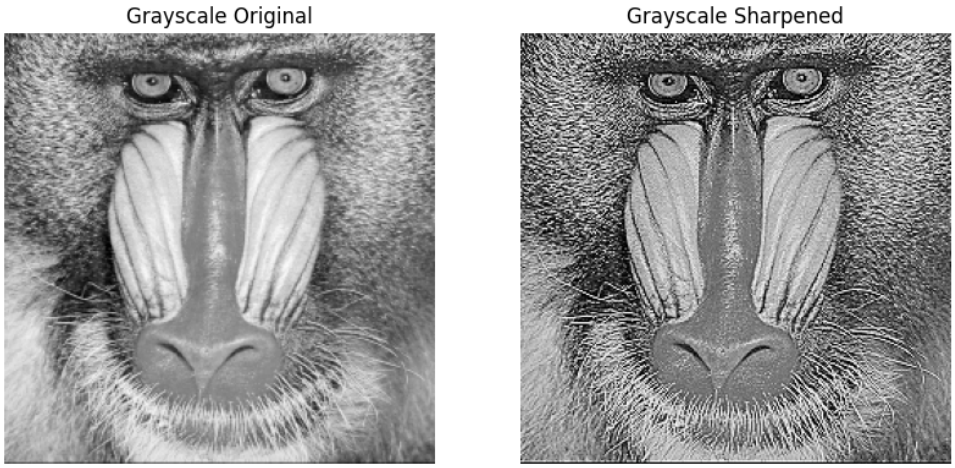

Figure 4: Original grayscale image and grayscale image after sharpening convolution using a 3×3 kernel. Task 6

Table 6: 5×5 grayscale patch from the sharpened image (rows 90–94, columns 90–94). Task 6

179	200	186	193	178
197	192	179	190	167
151	209	198	211	187
177	164	185	217	183
206	176	190	191	212

8 Task 7: Convolution from scratch NumPy

In this task, two-dimensional convolution was implemented manually using NumPy in order to better understand the underlying operation of convolution. The same 3×3 blur kernel used in Task 5 was applied.

Zero padding was added to the input image so that the spatial resolution of the output image remained 256×256 pixels. For each pixel location, the kernel was multiplied element-wise with the corresponding local neighborhood, and the resulting values were summed to produce the output pixel value.

The manually computed convolution result was compared with the result obtained using the OpenCV function `cv2.filter2D`. Both images were displayed side by side to visually verify that the manual implementation produces the same smoothing effect as the built-in OpenCV function.

To further validate the correctness of the manual convolution, a 5×5 grayscale patch from rows 90 to 94 and columns 90 to 94 was extracted from both the manually blurred image and the OpenCV blurred image. The numerical values of the two patches were printed and compared.

Table 7: 5×5 grayscale patch obtained using manual convolution with a blur kernel (rows 90–94, columns 90–94). Task 7

180	185	187	185	183
180	185	188	187	185
180	185	189	189	188
181	185	189	191	190
182	183	187	190	190

Table 8: 5×5 grayscale patch obtained using OpenCV `filter2D` with a blur kernel (rows 90–94, columns 90–94). Task 7

181	185	187	185	184
181	185	189	187	186
181	185	189	189	188
182	185	189	191	190
183	184	187	191	191

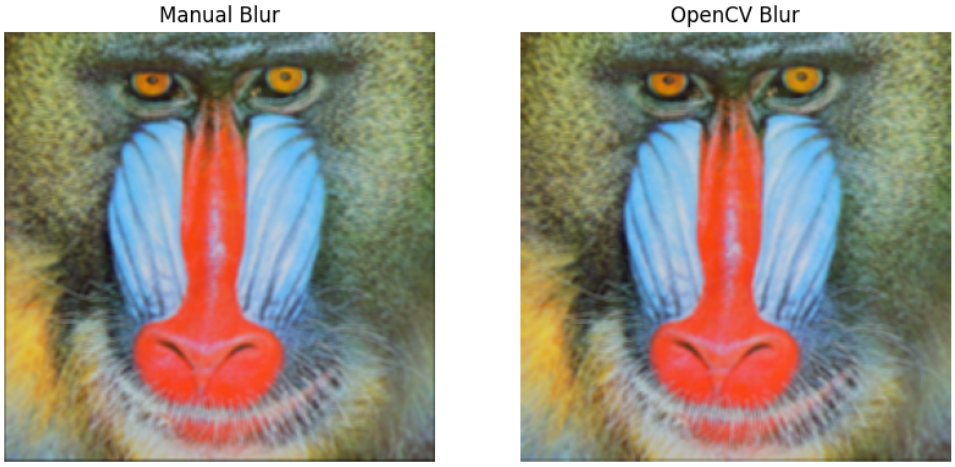

Figure 5: Comparison of grayscale images obtained using manual blur convolution and OpenCV `filter2D` blur. Task 7

9 Task 8: Maxpooling NumPy

In this task, maxpooling was implemented manually using NumPy to demonstrate one of the core operations used in convolutional neural networks. The input for maxpooling was

the blurred grayscale image obtained in Task 5.

Maxpooling was performed using a window size of 2×2 and a stride of 2. As a result, the spatial resolution of the image was reduced by a factor of two in both dimensions. The input image shape and the output image shape were printed to verify the dimensionality change.

To numerically illustrate the maxpooling operation, a 4×4 grayscale block from rows 90 to 93 and columns 90 to 93 of the input image was selected. The corresponding 2×2 pooled block was obtained from rows 45 to 46 and columns 45 to 46 of the output image, since the stride equals 2. The maximum value from each 2×2 window was selected to form the pooled output.

The shape of the input blurred grayscale image was printed as (256, 256). After applying maxpooling with a window size of 2×2 and a stride of 2, the output image shape became (128, 128), confirming the expected reduction of spatial resolution by a factor of two in both dimensions.

Table 9: 4×4 grayscale block from the blurred image used as input for maxpooling (rows 90–93, columns 90–93). Task 8

181	185	187	185
181	185	189	187
181	185	189	189
182	185	189	191

Table 10: 2×2 block obtained after maxpooling with window size 2×2 and stride 2 (rows 45–46, columns 45–46). Task 8

185	189
185	191

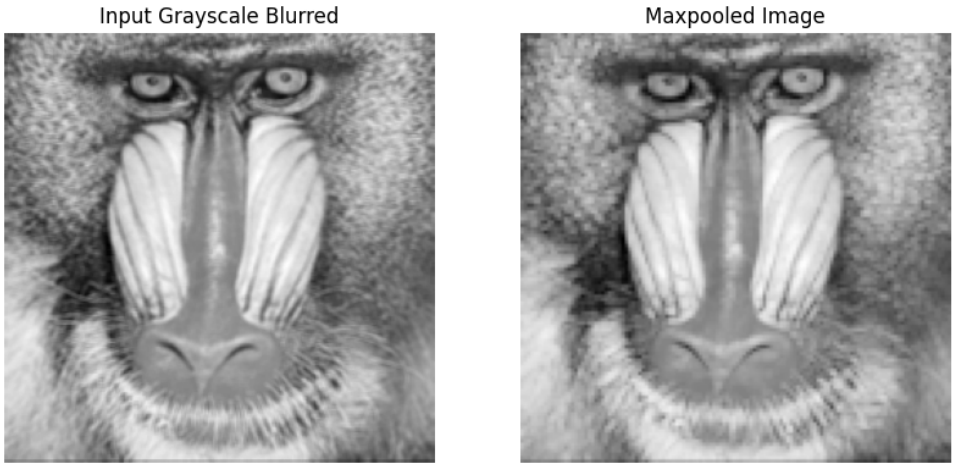

Figure 6: Input blurred grayscale image and the corresponding image after maxpooling with window size 2×2 and stride 2. Task 8

10 Task 9: Manual calculation of convolution

In this task, the convolution output value was computed manually for a given 3×3 input patch and a 3×3 kernel. No OpenCV or NumPy convolution functions were used.

The input patch was defined as:

$$\begin{bmatrix} 2 & 1 & 3 \\ 0 & 4 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

The convolution kernel was defined as:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Each element of the input patch was multiplied by the corresponding kernel element, and all results were summed:

$$2 \cdot 1 + 1 \cdot 1 + 3 \cdot 1 + 0 \cdot 1 + 4 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 + 2 \cdot 1 + 0 \cdot 1$$

$$= 2 + 1 + 3 + 0 + 4 + 2 + 1 + 2 + 0 = 15$$

Thus, the final convolution output value is 15.

11 Task 10: Manual calculation of maxpooling

In this task, maxpooling was computed manually for a given 4×4 input matrix using a window size of 2×2 and a stride of 2. No OpenCV or NumPy pooling functions were used.

The input matrix was defined as:

$$\begin{bmatrix} 3 & 1 & 4 & 2 \\ 0 & 5 & 2 & 1 \\ 6 & 2 & 1 & 7 \\ 4 & 3 & 0 & 5 \end{bmatrix}$$

The matrix was divided into four non-overlapping 2×2 windows:

Top-left window:

$$\begin{bmatrix} 3 & 1 \\ 0 & 5 \end{bmatrix} \Rightarrow \max = 5$$

Top-right window:

$$\begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \Rightarrow \max = 4$$

Bottom-left window:

$$\begin{bmatrix} 6 & 2 \\ 4 & 3 \end{bmatrix} \Rightarrow \max = 6$$

Bottom-right window:

$$\begin{bmatrix} 1 & 7 \\ 0 & 5 \end{bmatrix} \Rightarrow \max = 7$$

The final 2×2 maxpooled output matrix is:

$$\begin{bmatrix} 5 & 4 \\ 6 & 7 \end{bmatrix}$$

12 Results

The results of this laboratory work demonstrate the effect of various geometric transformations and image processing operations on digital images. The affine transformation successfully mapped selected points from the original image to their new positions, which was confirmed both numerically and visually.

The nonlinear geometric transformation implemented using sinusoidal remapping produced smooth spatial distortions across the image. Pixel value analysis showed that even when spatial coordinates remain the same, pixel intensities can differ due to remapping and interpolation.

Convolution with a blur kernel resulted in reduced local intensity variations by averaging neighboring pixels, while sharpening enhanced edges and fine details by increasing intensity differences. The manual convolution implementation produced results that closely matched the output of the OpenCV `filter2D` function, confirming the correctness of the manual approach.

Maxpooling reduced the spatial resolution of the image while preserving the most significant intensity values. The numerical examples clearly illustrated how maximum values are selected from local neighborhoods. Manual calculations of convolution and maxpooling further reinforced the understanding of these operations at a fundamental level.

13 Conclusion

In this laboratory work, basic geometric transformations and core CNN operations were studied and implemented using OpenCV and NumPy. Affine and nonlinear transformations demonstrated how image geometry can be modified through matrix-based and pixel-wise mappings.

Convolution operations showed how local image features can be smoothed or enhanced depending on the applied kernel. The manual implementation of convolution and maxpooling provided deeper insight into how these operations function internally and how they affect pixel values and image structure.

Overall, the laboratory work demonstrated that image preprocessing plays a crucial role in computer vision and deep learning pipelines. The combination of theoretical understanding, numerical verification, and visual analysis helps build a strong foundation for further study of convolutional neural networks and advanced computer vision methods.

14 References

- OpenCV Documentation: `https://docs.opencv.org`
- Computer Vision lectures by Koishiyeva Dina
- Google Colab Notebook: `https://colab.research.google.com/drive/1wYsMhW7DV58jBErw0`
  `usp=sharing`
- Google Drive as repository: `https://drive.google.com/drive/folders/17ApmnB7akOcHhGw0`
  `usp=sharing`

A Appendix

Additional screenshots, processed images, and outputs demonstrating the work results are provided in this section.

Figure 7: Original image used in the laboratory work

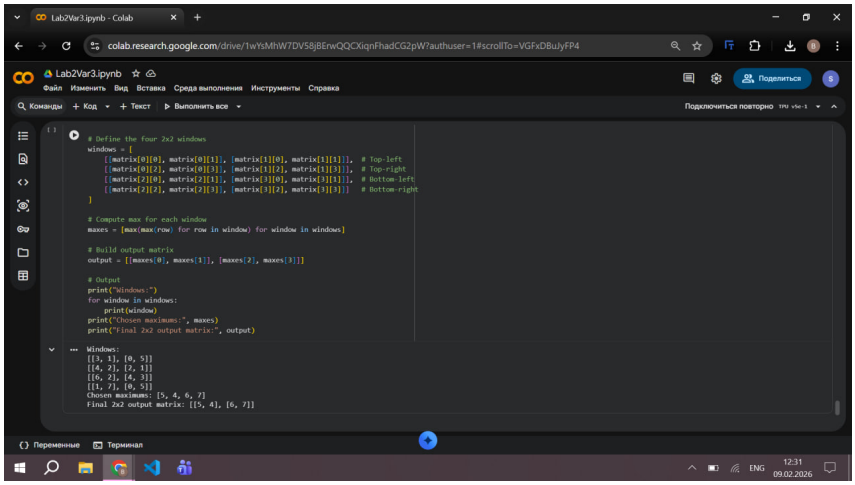

Figure 8: Proof that I did this lab work myself