AFFINE TRANSFORMATION

-Drishya Karki

AI and ML intern

In this exercise, I have implemented affine transformation on images. The transformations that are carried out in this project are

- Translation
- Rotation
- Shearing
- Horizontal Reflection
- Vertical Reflection

from scratch in python.

To achieve these objectives, I studied the basic mathematics behind them. So, let's go through all the mathematics involved in these transformations one by one:

Translation

Translation involves moving each point of the image by a specified distance in a specific direction. The matrix used for the translation along the x-axis and y-axis by certain distance tx and ty respectively is

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Here a 2-vector(x, y) is represented as a 3-vector(x, y, 1).

Rotation

It involves rotating the image by certain angle θ. The matrix used for counter-clockwise rotation through angle θ is

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Shearing

It is the transformation in which the shape of an object is distorted along one axis while keeping the other axis unchanged. There are two types of shearing: horizontal(along x-axis) and vertical(along y-axis). It is described using shearing factors Shx, Shy.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Vertical Reflection

It is a transformation in which an image or object is flipped along the vertical axis. In simpler words, the top part becomes the bottom part and the bottom part becomes the top part of an image or object.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Horizontal Reflection

It is a transformation in which an image or object is flipped along the horizontal axis. In simpler words, the right part becomes the left part and the left part becomes the right part of an image or object.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

So how to implement these in codes without using any geometric transformation libraries?

So, now to implement these mathematical calculations in the basic level, I did the following steps:

- First of all I needed to take the image as input and then convert it into an array for performing the desired transformation. So to take image as input, I used PILLOW and to convert the image into array I used NumPy.
- Then I wrote a function for matrix multiplication: matrix_multiplication(matrix, vector) which takes two arguments as you can see.
- After that I also wrote a function new_arr(arr) to create a new array filled with zeros, so that it is of the same dimensions as that of our original image but is empty right now.
- Out of all the affine transformations, translation was performed at first. The function for translation is written purely with basic matrix multiplications and operations without the use of any external libraries. The apply_translation(image, tx, ty), that I wrote, returns the translated_image which is shown in the code later in the document. Finally the translated_image which is returned and is an array is converted into the corresponding image and the final translated image is given back to the user.
- Similarly, rotations, shearing, horizontal and vertical reflections were performed using basic matrix operations without the use of any external libraries.
- Once the transformed arrays were returned from various functions, they were converted into images and given back to the user.

Code

Matrix multiplication and new array filled with zero

```python
def matrix_multiplication(matrix, vector):
    You, 1 second ago • Uncommitted changes
    result = [0, 0, 0]

    for i in range(3):
        for j in range(3):
            result[i] += matrix[i][j] * vector[j]

    return result


def new_arr(array):

    shape = array.shape
    dtype = array.dtype

    zeros_array = np.zeros(shape, dtype)

    return zeros_array
```

Translation function

```python
def apply_translation(image, tx, ty):

    image_array = np.array(image)

    rows, cols, channels = image_array.shape

    translated_image = new_arr(image_array)
    translation_matrix = np.array([[1, 0, tx],
                                   [0, 1, ty],
                                   [0, 0, 1]])
    You, yesterday • Add Image Translation Functionality …
    for i in range(rows):
        for j in range(cols):
            translated_coords = matrix_multiplication(translation_matrix, [j, i, 1])

            translated_x = int(translated_coords[0])
            translated_y = int(translated_coords[1])

            if 0 <= translated_x < cols and 0 <= translated_y < rows:
                translated_image[translated_y, translated_x] = image_array[i, j]

    return translated_image
```

Rotation Function

```python
def apply_rotation(image, angle_degree):

    rads = math.radians(angle_degree)
    cx, cy = (image.width // 2, image.height // 2)

    height_rot_img = round(abs(image.height * math.sin(rads))) + round(abs(image.width * math.cos(rads)))
    width_rot_img = round(abs(image.width * math.cos(rads))) + round(abs(image.height * math.sin(rads)))

    rot_img = Image.new("RGB", (width_rot_img, height_rot_img))

    midx, midy = (width_rot_img // 2, height_rot_img // 2)

    for i in range(rot_img.height):
        for j in range(rot_img.width):

            x = (i - midx) * math.cos(rads) + (j - midy) * math.sin(rads)
            y = -(i - midx) * math.sin(rads) + (j - midy) * math.cos(rads)
            x = round(x) + cy
            y = round(y) + cx

            if 0 <= x < image.height and 0 <= y < image.width:
                rot_img.putpixel((j, i), image.getpixel((y, x)))

    return rot_img
```

Shearing Function

```python
def apply_shearing(image, shear_x, shear_y):

    image = np.array(image)

    rows, cols, channels = image.shape

    new_cols = int(cols + abs(shear_x) * rows)
    new_rows = int(rows + abs(shear_y) * cols)

    sheared_image = np.zeros((new_rows, new_cols, channels), dtype = np.uint8)

    for i in range(rows):
        for j in range(cols):       You, 2 hours ago • Implemented Shearing Tran
            sheared_x = int(j + shear_x * i)
            sheared_y = int(i + shear_y * j)

            if 0 <= sheared_x < new_cols and 0 <= sheared_y < new_rows:
                sheared_image[sheared_y, sheared_x] = image[i, j]

    return sheared_image
```

Vertical and Horizontal Reflection Function

```
def apply_vertical_reflection(image):

    width, height = image.size
    reflected_image = Image.new("RGB", (width, height))

    for y in range(height):
        for x in range(width):
            original_pixel = image.getpixel((x, y))
            reflected_x = x
            reflected_y = height - y - 1
            reflected_image.putpixel((reflected_x, reflected_y), original_pixel)

    return reflected_image


def apply_horizontal_reflection(image):

    width, height = image.size
    reflected_image = Image.new("RGB", (width, height))

    for y in range(height):
        for x in range(width):
            original_pixel = image.getpixel((x, y))
            reflected_x = width - x - 1
            reflected_y = y
            reflected_image.putpixel((reflected_x, reflected_y), original_pixel)

    return reflected_image
```

Input and Output

```
image = Image.open("check.png")

rotated_image = apply_rotation(image, 45)
rotated_image.show()

translated_image = apply_translation(image, 100, 100)
translated_image = Image.fromarray(translated_image)
translated_image.show()

sheared_image = Image.fromarray(apply_shearing(image, 0.3, 0.5))
sheared_image.show()

vertical_reflected_image = apply_vertical_reflection(image)
vertical_reflected_image.show()

horizontal_reflected_image = apply_horizontal_reflection(image)
horizontal_reflected_image.show()
```

So, these all are the codes which are used in this project.

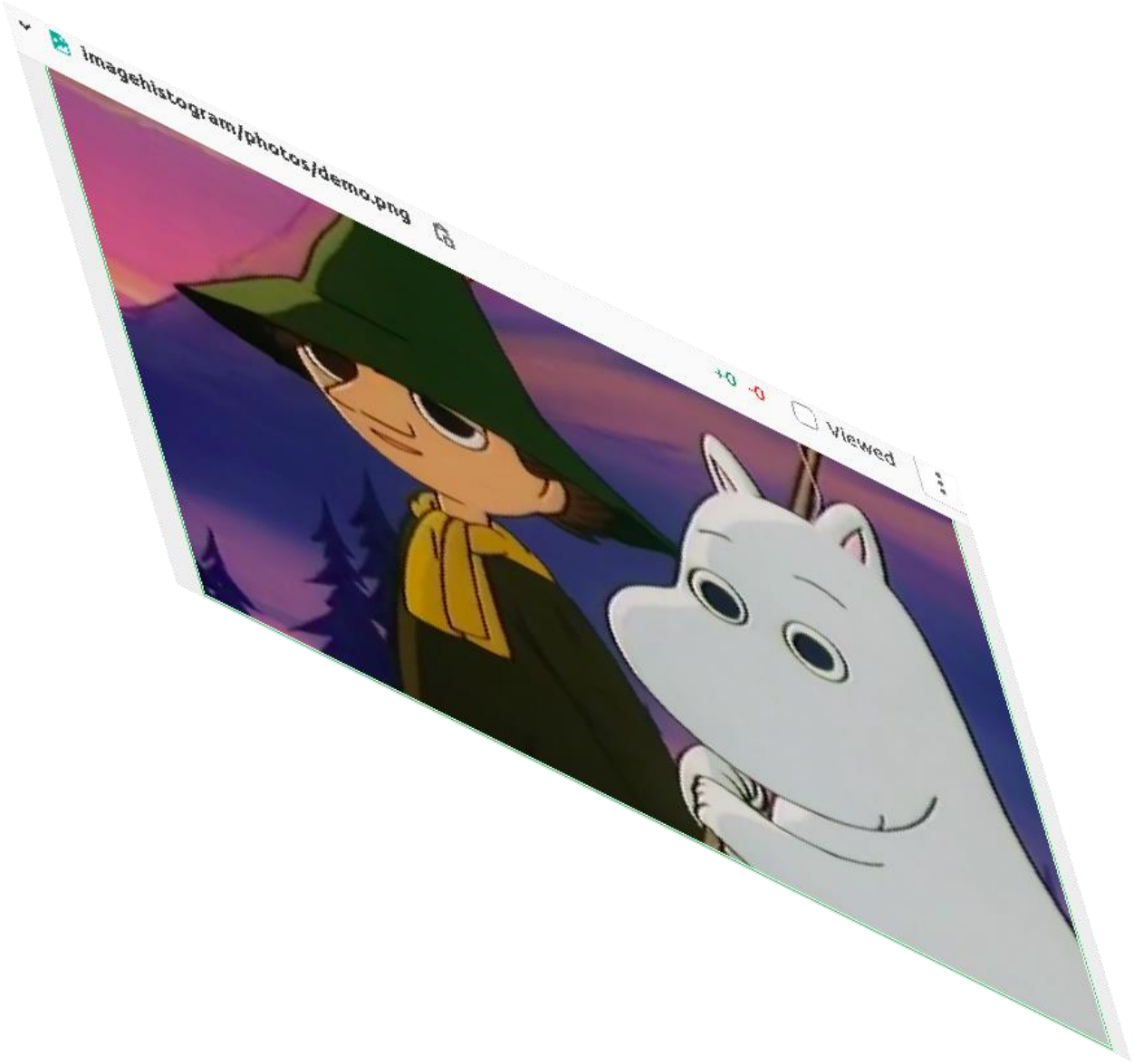
The image which is used for testing is

And the different outputs as per the codes are:

Rotation

Translation

Shearing



Vertical Reflection

Horizontal Reflection

Conclusion

In this way, I learned a lot about affine transformation and how they are performed in the basic level. I learned about the importance of preserving the dimension while working with the images because while transforming the images, at the beginning some of the data were lost. And hence I was aware about these qualities. And of course the matrix multiplications and other basic operations, working with trigonometric functions, how to play with them and how to convert images into array, perform the necessary operations and then again convert them back to the image format. These all things were learnt in this exercise.

-Drishya Karki

AI and ML intern