

Assignment: Image Formation

NAME: Sneha Dey ; I'D:221003003010

1: Image Formation

- Objective: Understand how geometric transformations affect images.
- Instructions:
 1. Write a function to perform translation, rotation, and scaling on an image without using any library functions.
 2. Use a sample image and apply each transformation.
 3. Display the original and transformed images side by side.

Code:

```
jupyter Untitled14 Last Checkpoint: 08/01/2022 (unsaved changes) Python 3 (ipykernel)
```

```
File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)
```

```
In [8]: 1 import numpy as np
2 from PIL import Image
3 import requests
4 from io import BytesIO
5 def translate(image, tx, ty):
6     width, height = image.size
7     translated_image = Image.new("RGB", (width, height))
8     pixels_old = image.load()
9     pixels_new = translated_image.load()
10    for x in range(width):
11        for y in range(height):
12            new_x = x + tx
13            new_y = y + ty
14            if 0 <= new_x < width and 0 <= new_y < height:
15                pixels_new[new_x, new_y] = pixels_old[x, y]
16    return translated_image
17 def rotate(image, angle):
18     angle_rad = np.radians(angle)
19     width, height = image.size
20     rotated_image = Image.new("RGB", (width, height))
21     center_x, center_y = width // 2, height // 2
22     pixels_old = image.load()
23     pixels_new = rotated_image.load()
24     for x in range(width):
25         for y in range(height):
26             new_x = int((x - center_x) * np.cos(angle_rad) - (y - center_y) * np.sin(angle_rad) + center_x)
27             new_y = int((x - center_x) * np.sin(angle_rad) + (y - center_y) * np.cos(angle_rad) + center_y)
28             if 0 <= new_x < width and 0 <= new_y < height:
29                 pixels_new[new_x, new_y] = pixels_old[x, y]
30     return rotated_image
31 def scale(image, scale_factor):
32     width, height = image.size
33     new_width = int(width * scale_factor)
34     new_height = int(height * scale_factor)
35     scaled_image = Image.new("RGB", (new_width, new_height))
36     pixels_old = image.load()
37     pixels_new = scaled_image.load()
38     for x in range(new_width):
39         for y in range(new_height):
40             old_x = int(x / scale_factor)
41             old_y = int(y / scale_factor)
42             if old_x < width and old_y < height:
43                 pixels_new[x, y] = pixels_old[old_x, old_y]
44     return scaled_image
45
46
47
```

```
jupyter Untitled14 Last Checkpoint: 08/01/2022 (unsaved changes) Python 3 (ipykernel)
```

```
File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)
```

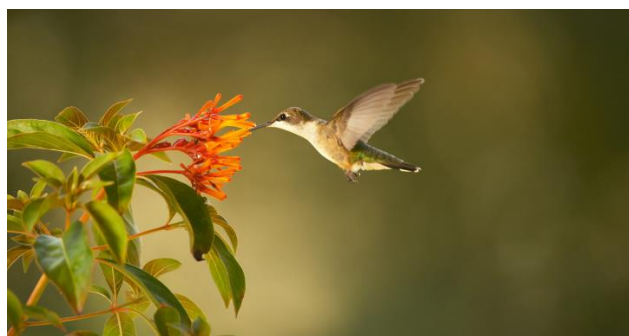
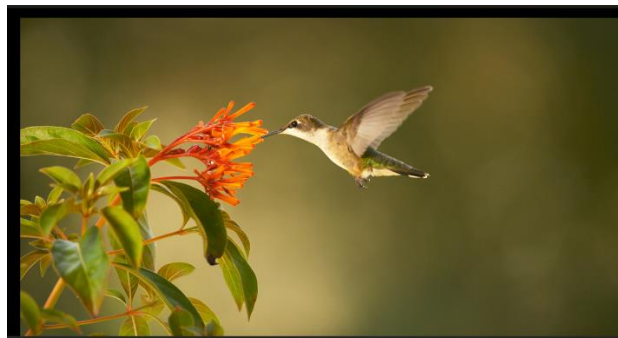
```
45 def load_image_from_url(url):
46     response = requests.get(url)
47     img = Image.open(BytesIO(response.content))
48     return img
49 # Display image in separate windows
50 def display_images_in_windows(images, titles):
51     for img, title in zip(images, titles):
52         img.show(title=title)
53 url = "https://www.pbs.org/wnet/nature/files/2019/07/Super-Hummingbirds-2.jpg"
54 original_image = load_image_from_url(url)
55 translated_image = translate(original_image, 50, 30)
56 rotated_image = rotate(original_image, 180)
57 scaled_image = scale(original_image, 0.5)
58 display_images_in_windows(
59     [original_image, translated_image, rotated_image, scaled_image],
60     ["Original Image", "Translated Image", "Rotated Image", "Scaled Image"]
61 )
```

OUTPUT:

Original image:



Transformed images:



2: Photometric Models

- Objective: Understand how lighting conditions affect pixel intensities.
- Instructions:
 1. Write a function to simulate different lighting conditions (e.g., increase brightness, decrease brightness).
 2. Apply these changes to a sample image.
 3. Display the original and altered images.

Code:

```
jupyter Untitled14 Last Checkpoint: 08/01/2022 (unsaved changes) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

In [23]: 1 from PIL import Image
2 import numpy as np
3 import requests
4 from io import BytesIO
5 def increase_brightness(image, value):
6     img_array = np.array(image)
7     bright_img = np.clip(img_array + value, 0, 255).astype(np.uint8)
8     return Image.fromarray(bright_img)
9 def decrease_brightness(image, value):
10    img_array = np.array(image)
11    dim_img = np.clip(img_array - value, 0, 255).astype(np.uint8)
12    return Image.fromarray(dim_img)
13 def load_image_from_url(url):
14    response = requests.get(url)
15    img = Image.open(BytesIO(response.content))
16    return img
17 def display_images_side_by_side(images):
18    widths, heights = zip(*(i.size for i in images))
19    total_width = sum(widths)
20    max_height = max(heights)
21    combined_image = Image.new('RGB', (total_width, max_height))
22    x_offset = 0
23    for img in images:
24        combined_image.paste(img, (x_offset, 0))
25        x_offset += img.size[0]
26    combined_image.show()
27 url="https://img.freepik.com/free-photo/wide-angle-shot-single-tree-growing-clouded-sky-during-sunset-surrounded-by-
28    "grass_181624-22807.jpg"
29 original_image = load_image_from_url(url)
30 brighter_image = increase_brightness(original_image, 50)
31 darker_image = decrease_brightness(original_image, 50)
32 display_images_side_by_side([original_image, brighter_image, darker_image])
33
```

OUTPUT:



3: Sampling and Quantization

- Objective: Understand the effects of sampling and quantization on image quality.
- Instructions:
 1. Write a function to downsample and upsample an image.
 2. Write a function to quantize an image to different levels (e.g., 2-bit, 4-bit).
 3. Apply these functions to a sample image.
 4. Display the results on image quality.

CODE:

```
jupyter Untitled14 Last Checkpoint: 08/01/2022 (unsaved changes) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Help Trusted

In [25]: 1 from PIL import Image
2 import numpy as np
3 import requests
4 from io import BytesIO
5 def downsample(image, factor):
6     width, height = image.size
7     img_array = np.array(image)
8     downsampled_img = img_array[::factor, ::factor]
9     return Image.fromarray(downsampled_img)
10 def upsample(image, factor):
11     img_array = np.array(image)
12     upsampled_img = np.repeat(np.repeat(img_array, factor, axis=0), factor, axis=1)
13     return Image.fromarray(upsampled_img)
14 def quantize(image, bit_depth):
15     img_array = np.array(image)
16     levels = 2 ** bit_depth
17     scale_factor = 256 // levels
18     quantized_img = (img_array // scale_factor) * scale_factor
19     return Image.fromarray(quantized_img)
20 def load_image_from_url(url):
21     response = requests.get(url)
22     img = Image.open(BytesIO(response.content))
23     return img
24 def display_images_side_by_side(images):
25     widths, heights = zip(*(i.size for i in images))
26     total_width = sum(widths)
27     max_height = max(heights)
28     combined_image = Image.new('RGB', (total_width, max_height))
29     x_offset = 0
30     for img in images:
31         combined_image.paste(img, (x_offset, 0))
32         x_offset += img.size[0]
33     combined_image.show()
34 url = "https://img.freepik.com/free-photo/wide-angle-shot-single-tree-growing-clouded-sky-during-sunset-surrounded-by-
35     grass_181624-22807.jpg"
36 original_image = load_image_from_url(url)
37 downsampled_image = downsample(original_image, 4)
38 upsampled_image = upsample(downsampled_image, 4)
39 quantized_image_2bit = quantize(original_image, 2)
40 quantized_image_4bit = quantize(original_image, 4)
41 display_images_side_by_side([original_image, downsampled_image, upsampled_image, quantized_image_2bit, quantized_image_4bit])
42
```


OUTPUT:



4: Image Definition and Neighbourhood Metrics

- Objective: Understand how images are defined and represented, and explore neighborhood metrics.
- Instructions:
 1. Define an image as a 2D matrix and implement functions to compute basic neighborhood metrics (e.g., mean, median, standard deviation within a neighborhood).
 2. Apply these metrics to a sample image.
 3. Display the results and discuss their significance.

CODE:

```
jupyter Untitled14 Last Checkpoint: 08/01/2022 (autosaved)  Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

In [28]: 1 import numpy as np
2 from PIL import Image
3 import requests
4 from io import BytesIO
5 from scipy.ndimage import generic_filter
6 def load_image_from_url(url):
7     response = requests.get(url)
8     img = Image.open(BytesIO(response.content)).convert("L")
9     return np.array(img)
10 def compute_mean(image, size):
11     return generic_filter(image, np.mean, size=size)
12 def compute_median(image, size):
13     return generic_filter(image, np.median, size=size)
14 def compute_std(image, size):
15     return generic_filter(image, np.std, size=size)
16 def display_image(image_array, title="Image"):
17     img = Image.fromarray(image_array)
18     img.show(title=title)
19 url = "https://img.freepik.com/free-photo/wide-angle-shot-single-tree-growing-clouded-sky-during-sunset-surrounded-by-
20     "grass_181624-22807.jpg"
21 original_image = load_image_from_url(url)
22 neighborhood_size = (3, 3)
23 mean_image = compute_mean(original_image, neighborhood_size)
24 median_image = compute_median(original_image, neighborhood_size)
25 std_image = compute_std(original_image, neighborhood_size)
26 display_image(original_image, title="Original Image")
27 display_image(mean_image.astype(np.uint8), title="Mean Filtered Image")
28 display_image(median_image.astype(np.uint8), title="Median Filtered Image")
29 display_image(std_image.astype(np.uint8), title="Standard Deviation Filtered Image")
30
```

OUTPUT:

Original Image:



Mean Filtered Image:



Median Filtered Image:



Standard Deviation Filtered Image:



