Khulna University of Engineering & Technology

Course No: CSE4128
Course Name: Image Processing & Computer Vision

Project Name: Meme Generator

Submitted By
Sk. Azraf Sami
Roll: 1907115

Submitted To

Dr. Sk. Md. Masudul Ahsan
Professor
CSE, KUET

Dipannita Biswas
Lecturer
CSE, KUET

Date: 1th July 2024

Objectives:

1. **Image Processing and Computer Vision Demonstration:**

   To create a practical application that demonstrates fundamental techniques in image processing and computer vision.

2. **Text and Credit Addition:**

   To implement functionality for adding customized text and credits to images, allowing for the creation of personalized memes.

3. **Image Padding**:

   To provide the capability to add padding to images, facilitating proper formatting and alignment of meme content.

4. **Image Collage Creation**:

   To enable users to create collages by combining multiple images into a single composite image.

5. **Black-and-White Conversion**:

   To offer an option for converting images to black and white, highlighting the use of grayscale image processing techniques.

6. **Image Enhancement**:

   To enhance image quality using histogram equalization, demonstrating methods for improving contrast and visual appeal.

7. **Image Blurring:**

   To implement Gaussian blur, showcasing techniques for reducing image noise and smoothing.

8. **Background Removal**:

   To apply advanced techniques such as GrabCut, mask generation, and morphology operations for effective background removal from images.

9. **Interactive User Interface:**

   To develop an intuitive and interactive user interface using Flask, allowing users to apply various image processing techniques easily.

Introduction:

The meme-generator project is an innovative application developed for the Image Processing and Computer Vision laboratory. This project leverages the power of Python and various image processing libraries to create a versatile tool for generating and editing memes. Memes have become a ubiquitous form of communication in the digital age, combining humor and imagery to convey messages effectively. This project not only provides a practical application for meme creation but also serves as an educational platform for exploring fundamental concepts in image processing and computer vision.

The application is built using the Flask framework, offering a user-friendly web interface where users can perform various image manipulation tasks. Key features include adding custom text and credits to images, padding, creating collages, converting images to black and white, enhancing image quality through histogram equalization, applying Gaussian blur, and removing backgrounds using advanced techniques like GrabCut, mask generation, and morphological operations.

By integrating these functionalities, the meme-generator project demonstrates essential image processing techniques in a real-world context. The project also introduces interactive features that allow users to adjust text size and position dynamically, enhancing the overall user experience. Through this project, users gain hands-on experience with image processing tools and techniques, making it a valuable educational resource.

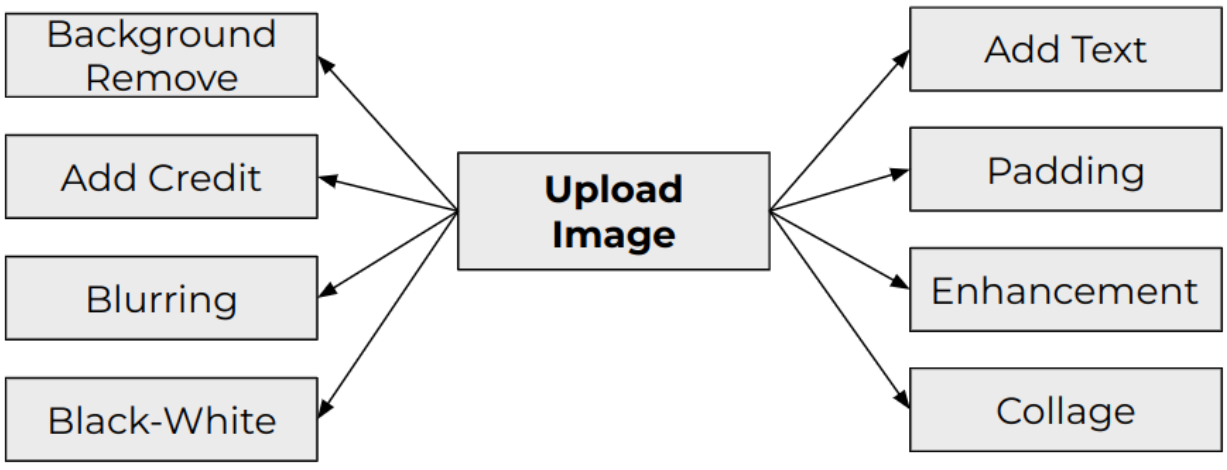

Fig-1 : Block Diagram of features

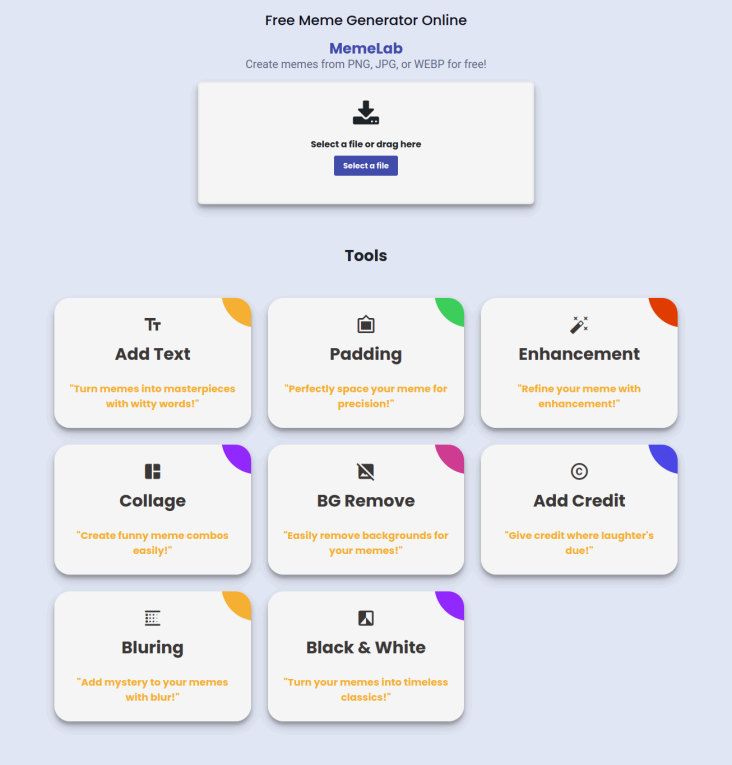

Fig-2: Home Page

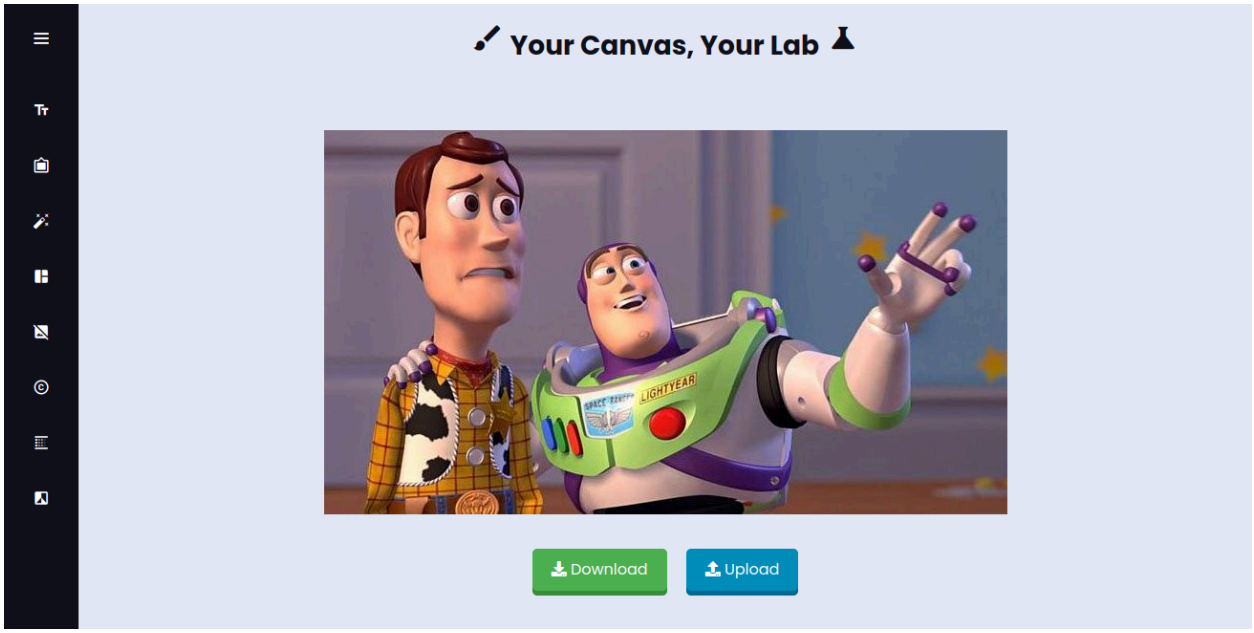

Fig-3: After uploading the image, tools page

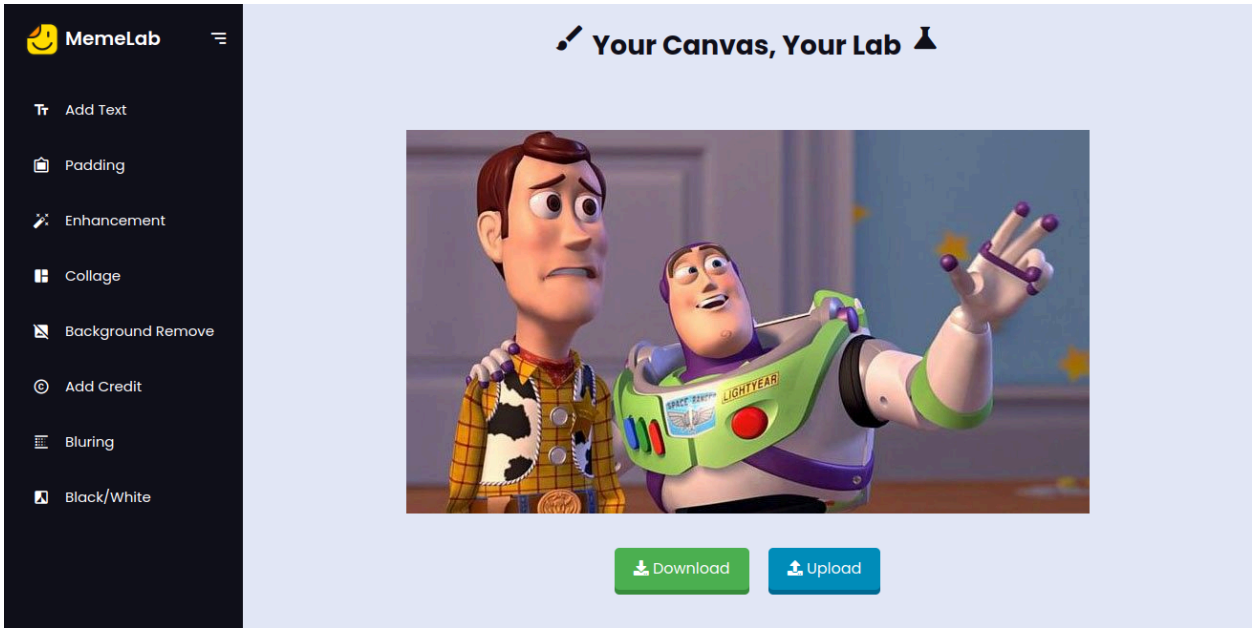

Fig-4: Tools page with tool name

Theory:

The meme-generator project is grounded in fundamental concepts of image processing and computer vision, which are essential for manipulating and enhancing digital images. This section explores the theoretical underpinnings of the key features implemented in the project.

Text Addition

☐ **Algorithm:**

1. **Initialize Flask Application**:
    - Configure upload folders.
2. **Define `add_text` Function**:
    - Open the image from the given path and convert it to RGBA mode.
    - Create a drawing context.
    - Load the specified font with the given size.
    - Parse the text position from JSON.
    - Draw the text at the specified position with the specified color and font.
    - Convert the image back to RGB mode.
    - Save the modified image to a specified path.
    - Return the path to the modified image

☐ **Pseudocode:**

Initialize Flask application

Configure upload folder and collage folder

Function add_text:

Open image from image_path and convert to RGBA

Create drawing context

Load font with specified font_size

Set text fill color to text_color with full opacity

Parse text_position from JSON

Extract text_x, text_y, text_width, text_height from parsed position

Draw text at (text_x, text_y) with fill_color and font

Convert image back to RGB

Save modified image to meme_path

Return meme_path

☐ **Block Diagram:**

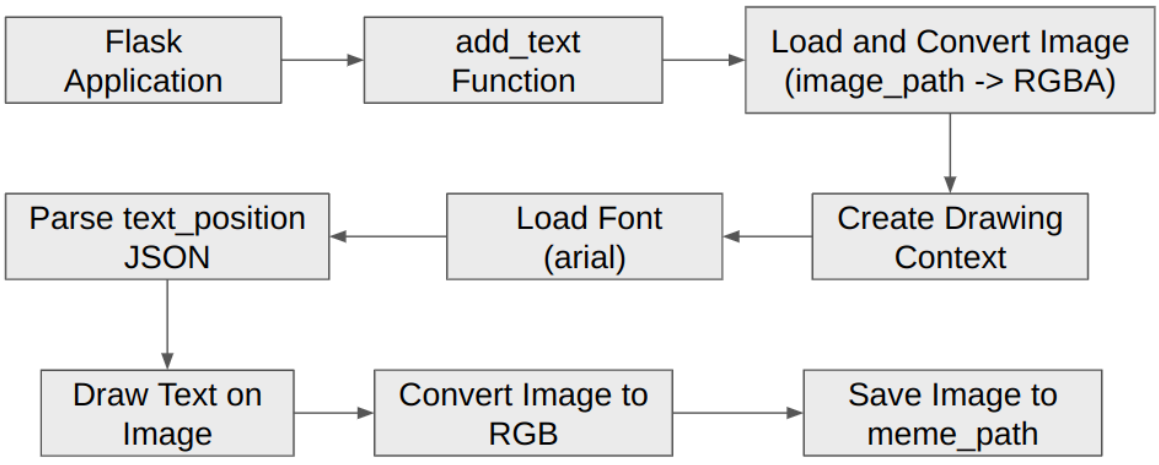

Fig-5: Block Diagram of Adding Text Tool

- [ ] **UI**

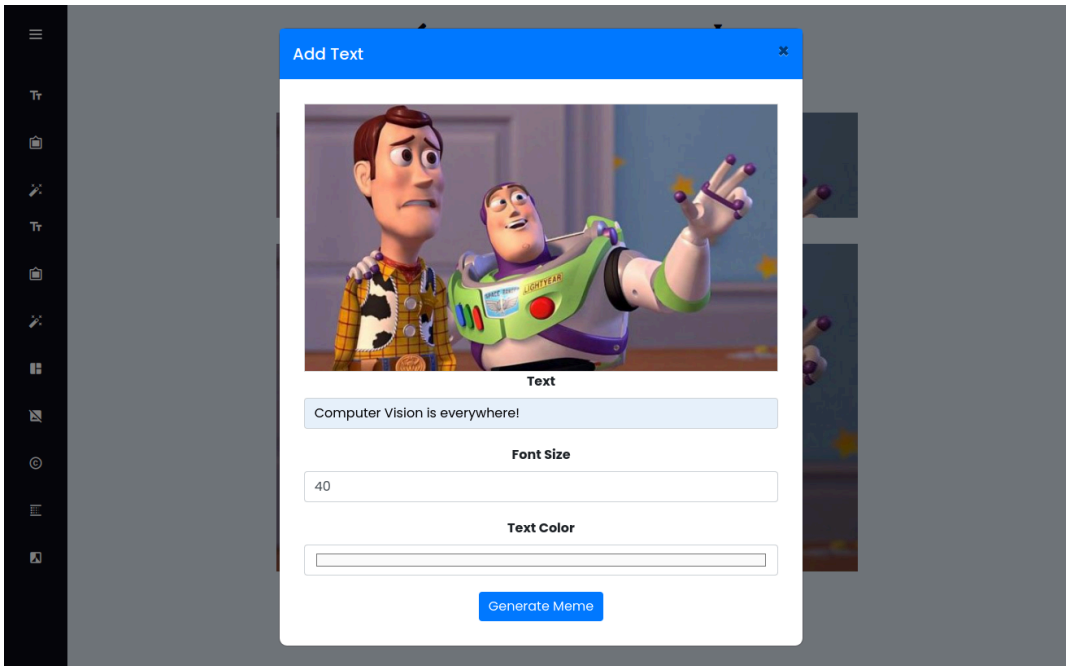

Fig-6: Adding Text Tool

Fig-7: Output Image

Padding

☐ Algorithm

Define `add_padding` Function:

- Open the image from the given path using PIL.
- Convert the PIL image to an OpenCV image (numpy array).
- Add padding to the image using the specified padding values and color.
- Convert the OpenCV image (numpy array) back to a PIL image.
- Save the padded image to a specified path.
- Return the path to the padded image

☐ Pseudocode

Function add_padding:
  Open image from image_path using PIL
  Convert PIL image to OpenCV image (numpy array)
  Add padding to image using cv2.copyMakeBorder with specified padding and color
  Convert OpenCV image (numpy array) back to PIL image
  Save padded image to meme_path
  Return meme_path

☐ Block Diagram

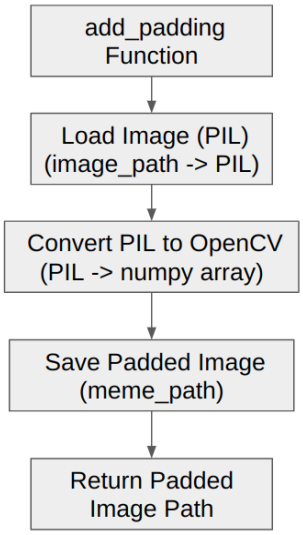

Fig-8: Block Diagram of Padding Function

☐ **UI**

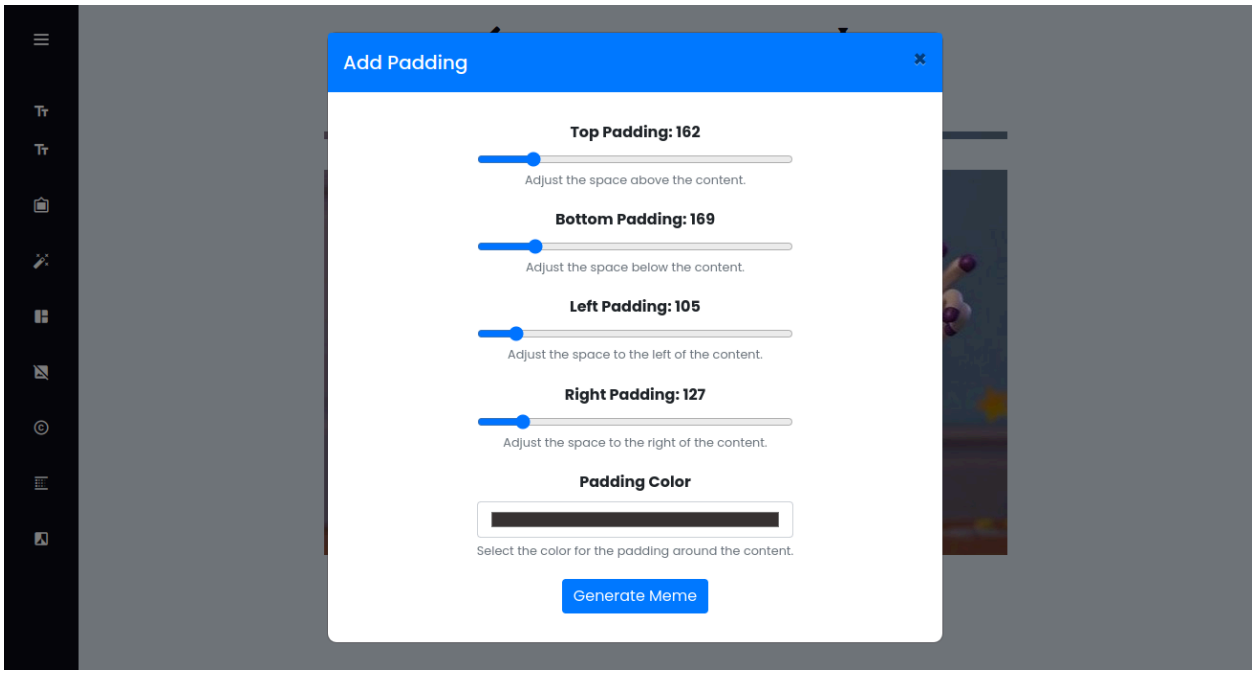

Fig-9: UI of Padding

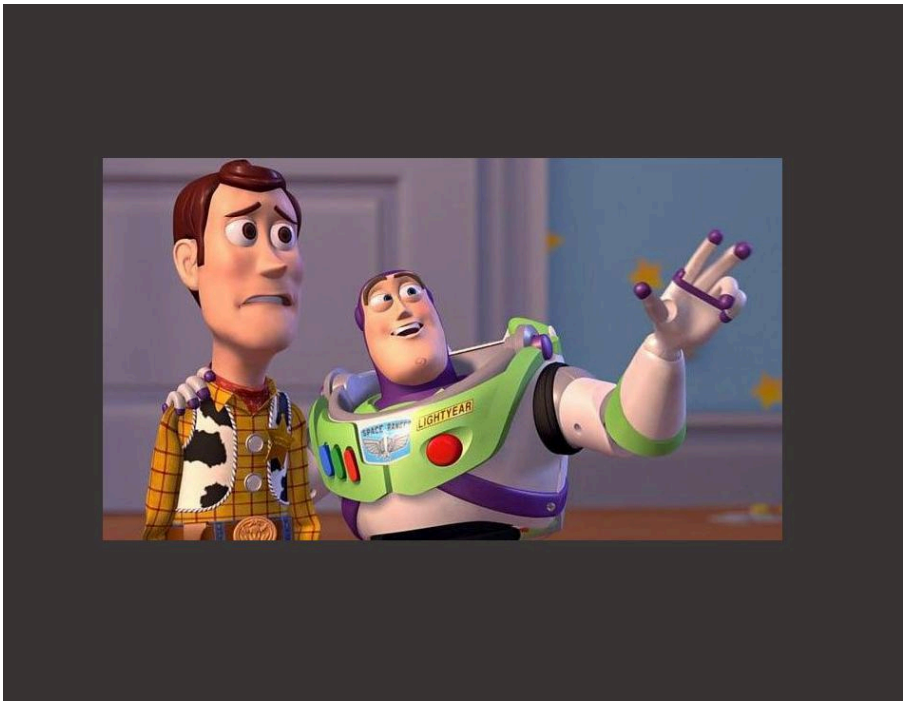

Fig-10: Output Image of Padding Tool

Enhancement

- [ ] **Algorithm**

Initialize Flask Application:

- Configure the upload folder and allow file extensions.

Define `allowed_file` Function:

- Check if the uploaded file has an allowed extension.

Define `histogram_equalization` Function:

- Open the image using PIL and convert it to RGB.
- Convert the PIL image to a NumPy array.
- Apply histogram equalization:
    - For grayscale images, compute the histogram, PDF, and CDF, then map the intensities using the CDF.
    - For color images, convert to YCrCb color space, apply histogram equalization to the Y channel, and merge channels back.
- Save the processed image to a specified path.
- Return the path to the processed image.

Define `upload_file` Route:

- Handle GET and POST requests.
- Validate and save uploaded files.
- Apply histogram equalization to the uploaded image.
- Render the result template with the processed image path

- [ ] **Pseudocode**

```
Initialize Flask application
Configure upload folder and allowed extensions

Function allowed_file:
    Check if file extension is allowed
    Return True if allowed, otherwise False

Function histogram_equalization:
    Open image from image_path and convert to RGB
    Convert image to NumPy array
    If grayscale image:
        Compute histogram, PDF, and CDF
        Map intensities using CDF
    Else (color image):
```

Convert to YCrCb color space
Apply histogram equalization to Y channel
Merge channels back
Convert back to RGB
Save processed image to meme_path
Return meme_path

Route upload_file (GET, POST):
If POST request:
Validate file part
Save uploaded file
Apply histogram equalization
Render result template with processed image path
Else:
Render upload template

Run Flask application in debug mode

- [ ] **Block Diagram**

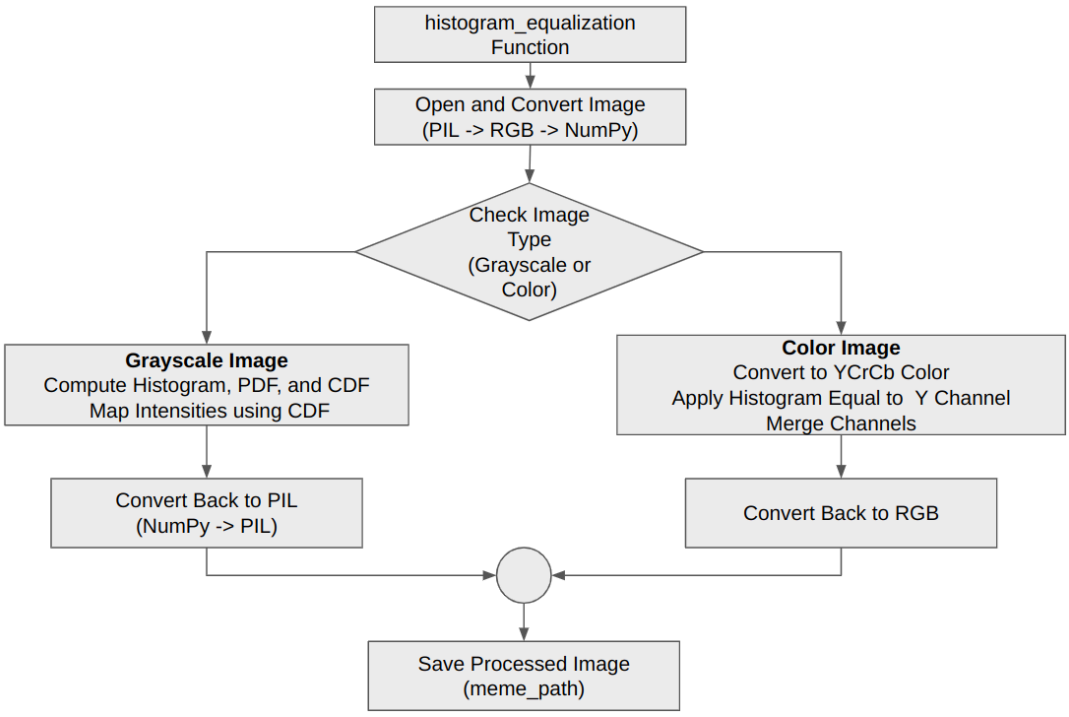

Fig-11: Block Diagram of Enhancement Tool

- [ ] **UI**

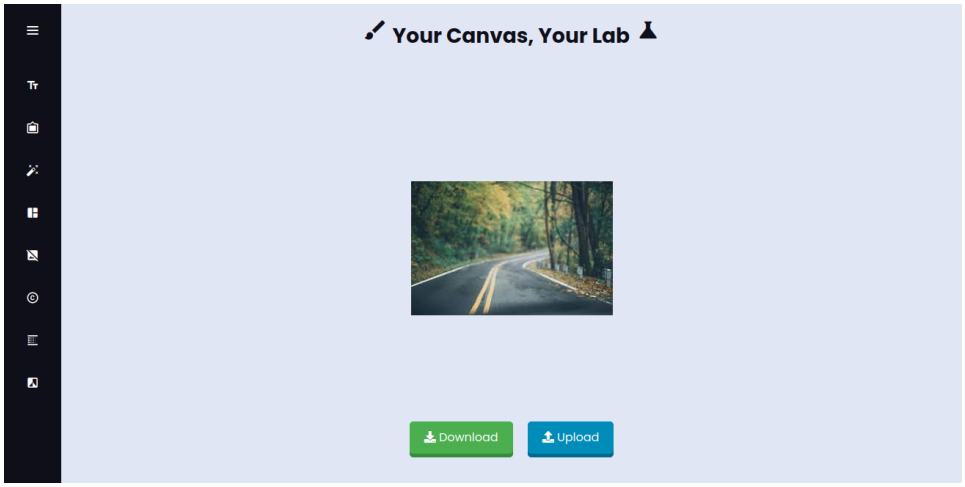

Fig-12: Input Image of Enhancement

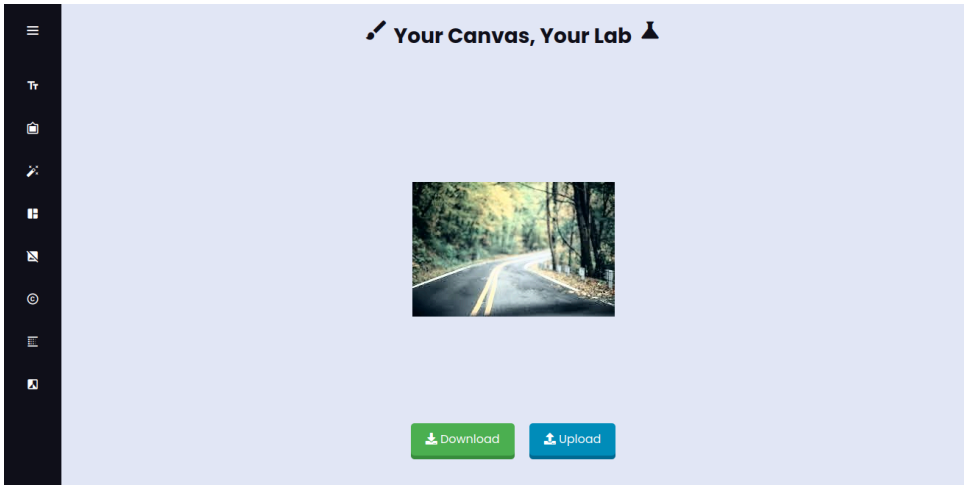

Fig-13: Output Image of Enhancement

Collage

- [ ] **Algorithm**

Define `collage` Function:

- Create a list of lists (`images`) containing paths to each image in the collage, organized by rows and columns.
- Calculate the dimensions (`collage_width` and `collage_height`) of the final collage based on the number of rows and the maximum number of images in any row.

- Create a new blank image (`collage`) using PIL, with dimensions calculated in the previous step and a white background.
- Resize each image to fit within its designated space in the collage, maintaining aspect ratio.
- Paste each resized image into its corresponding position in the collage.
- Save the completed collage image to a specified path (`meme_path`).

☐ **Pseudocode**

```
Function collage(rows, cells, images_path):

    # Create a list of lists of images

    images = [[os.path.join(images_path, f'{i}_{j}.jpg') for j in range(1, cells[i-1]+1)] for i in range(1, rows+1)]

    # Define the size of each image in the collage

    image_size_factor = 400

    # Determine the size of the collage

    collage_height = len(images) * image_size_factor

    collage_width = max(len(row) for row in images) * image_size_facto

    # Create a new image for the collage with white background

    collage = Image.new('RGB', (collage_width, collage_height), 'white')

    # Paste each image into the collage

    y = 0

    for row in images:

        x = 0

        img_width = collage_width // len(row)

        resized_imgs = [Image.open(i).resize((img_width, image_size_factor)) for i in row]

        for img in resized_imgs:

            collage.paste(img, (x, y))

            x += img_width
```

```
        y += image_size_facto

    # Save the collage

    meme_path = 'static/latest.jpg'

    collage.save(meme_path)

    return meme_path
```

☐ **Block Diagram**

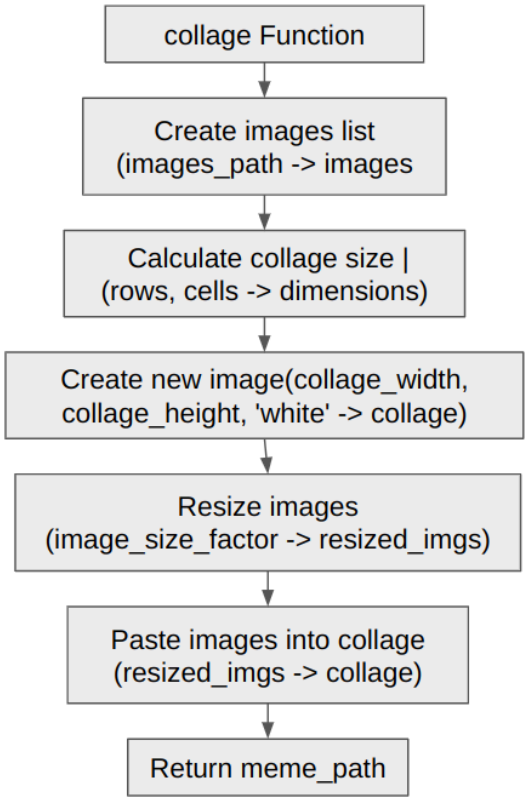

Fig-14: Block Diagram of Collage Tool

☐ UI

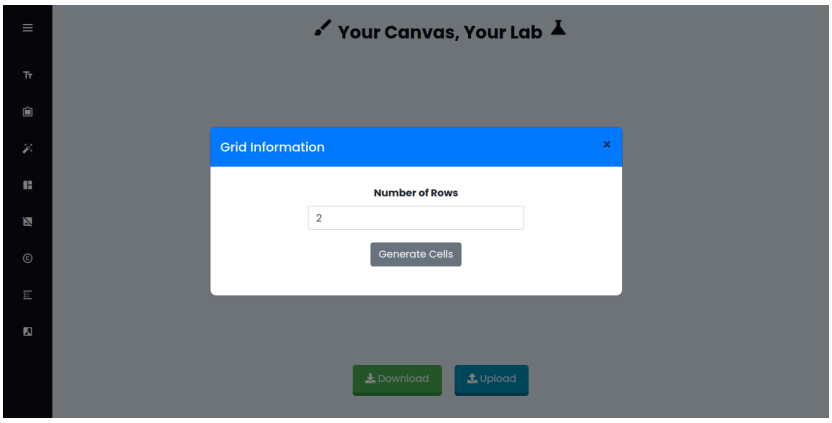

Fig-15: ‘Number of Rows’ Modal Box

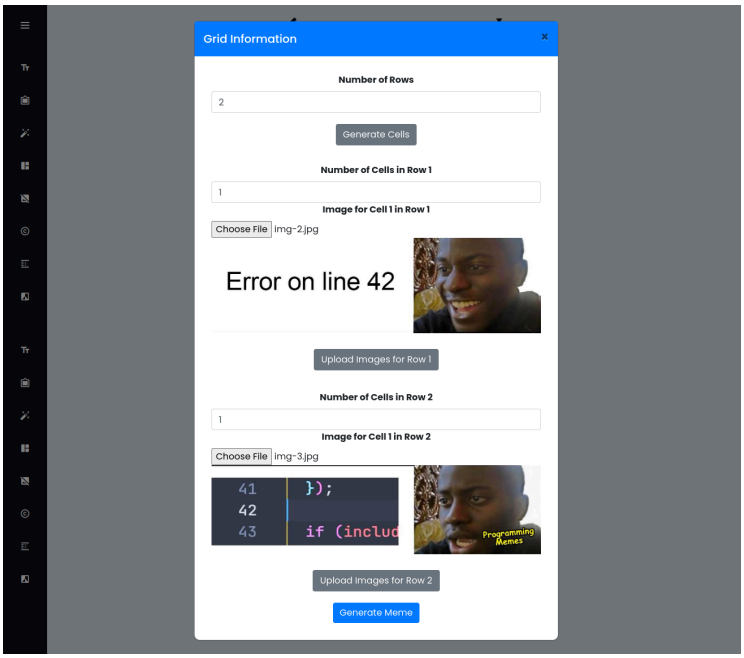

Fig-16: Taking Input Image For Each Row

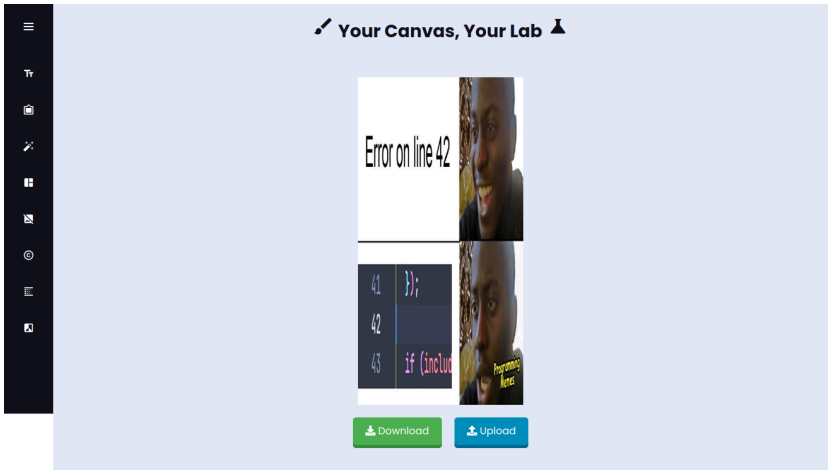

Fig-17: Output

Background Remove

☐ Algorithm

Import necessary libraries: Import `cv2`, `numpy`, and `PIL`.

Define the function `remove_background` which accepts an image path as input.

Define the internal function `custom_background_removal` to process the image:

- Convert the image to a NumPy array.
- Convert RGBA to RGB if necessary.
- Initialize the mask, background model, and foreground model for GrabCut.
- Define a rectangle around the image for initial segmentation.
- Apply GrabCut with increased iterations for better refinement.
- Create a binary mask for the foreground.
- Clean up edges using morphological operations.
- Add an alpha channel to create an RGBA image.
- Convert the processed NumPy array back to an image.

Open and load the image from the provided path.

Apply the custom background removal function to the loaded image.

Create a white background image of the same size.

Paste the processed image with transparency onto the white background.

Save the final image with a white background and return the output path

- [ ] **Pseudocode**

```
function remove_background(image_path):
    import necessary libraries (cv2, numpy, PIL)
    function custom_background_removal(image):
        convert image to numpy array (np_img)
        if image has RGBA channels:
            convert np_img from RGBA to RGB
        initialize mask, bgd_model, fgd_model for GrabCut
        define rectangle for initial segmentation
        apply GrabCut with increased iteration
        create binary mask for foreground
        apply mask to np_img
        clean up edges using morphological operations
        add alpha channel to np_img to create RGBA imag
        convert np_img back to PIL image
        return RGBA image
    open and load image from image_path
    apply custom_background_removal to the loaded image
    create white background image of the same size as processed image
    paste processed image onto white background
    save the final image with white background
    return output path
```

☐ Block Diagram

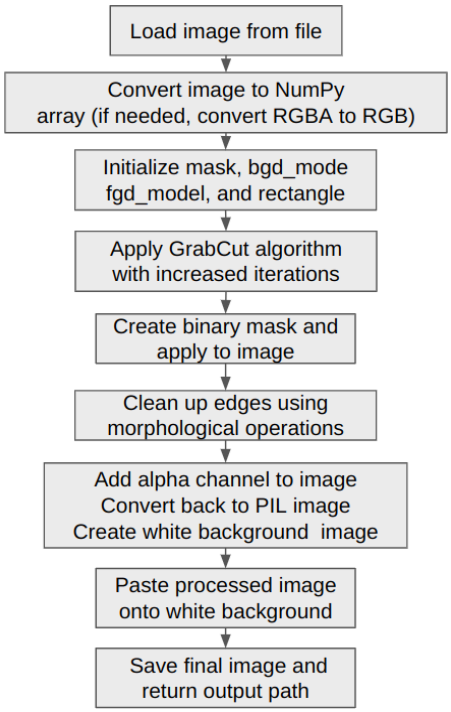

Fig-18: Block Diagram of Background Remove Tool

☐ UI

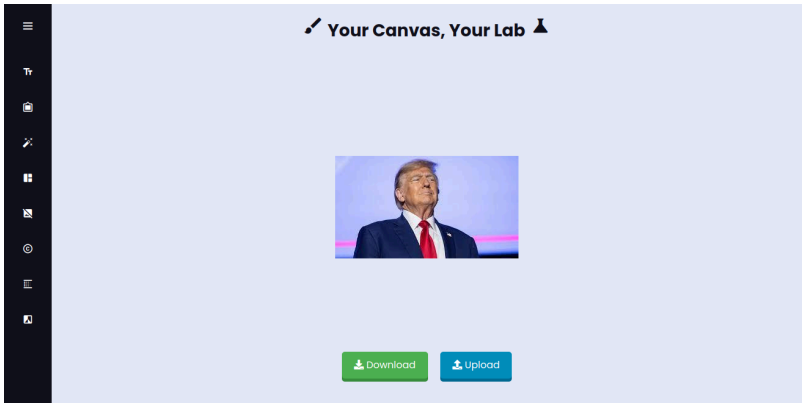

Fig-19: Input Image

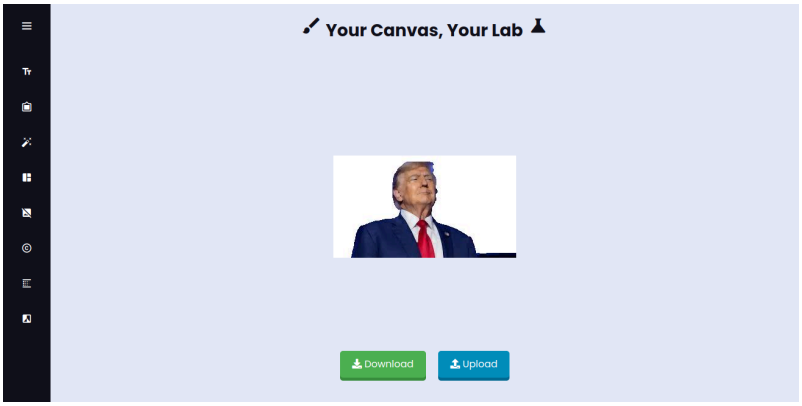

Fig-20: Output Image

Add Credit

☐ Algorithm

Import necessary libraries: Import `Image`, `ImageDraw`, `ImageFont` from PIL, and Flask-related modules.

Define the function `textsize` which calculates the size of the text using a specified font.

Define the function `add_credit` which adds watermark text to an image:

- Open the image and convert it to RGBA mode.
- Create a drawing context.
- Load the specified font with the given size.
- Calculate the size of the watermark text and its position on the image.
- Adjust font size if the text is too wide for the image.
- Determine the position for the watermark text (bottom-right corner).
- Set the color and opacity for the watermark text.
- Draw the watermark text multiple times with slight offsets to mimic boldness.
- Convert the image back to RGB mode before saving.
- Save the processed image and return its path.

☐ **Pseudocode**

function textsize(text, font):

create a new image in RGBA mode

create a drawing context

calculate text bounding box size

return width and height of the text

function add_credit(credit_text, font_size, opacity, text_color, boldness, image_path):

open the image and convert to RGBA mode

create a drawing context

load the font with specified size

calculate watermark text size and position

adjust font size if text is too wide for the image

determine position for watermark text (bottom-right corner)

set watermark text color and opacity

draw the watermark text multiple times with slight offsets to mimic boldness

convert the image back to RGB mode

save the processed image

return the path of the saved image

☐ Block Diagram

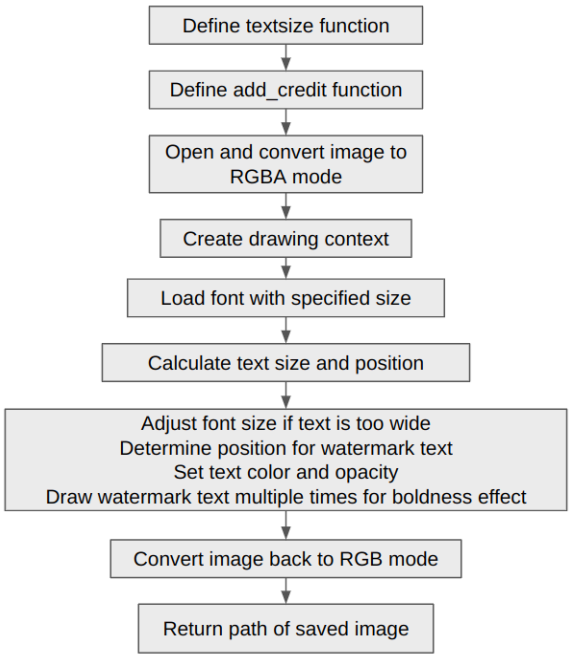

Fig:-21: Block Diagram of Add Credit Tool

☐ UI

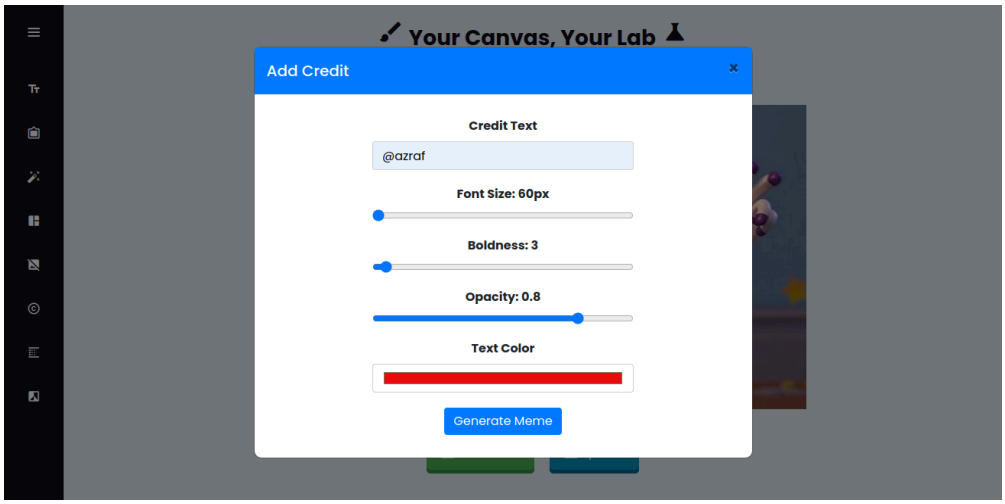

Fig-22: Modal Box for Add Credit

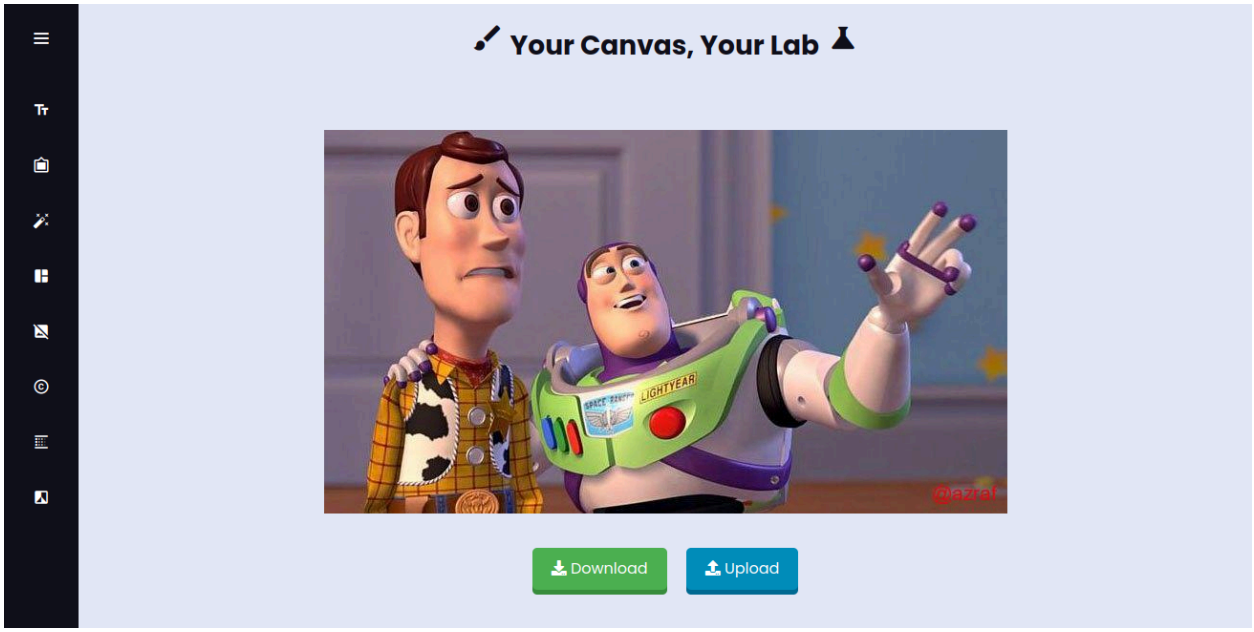

Fig-23: Output Image, after applying ‘Add Credit’ tool

Blurring

☐ Algorithm

Create Gaussian Kernel

- Generate a grid of coordinates.
- Compute the Gaussian function for each coordinate.
- Normalize the kernel so that the sum of all elements equals 1.

Apply Kernel to Image

- Normalize the image to the range [0, 1].
- Pad the image to handle borders using reflection padding.
- Convolve the kernel with the image.
- Clip the resulting values to the range [0, 255] and convert to 8-bit.

Apply Gaussian Blur

- Load the image and convert it to a NumPy array.
- Ensure the kernel size is odd.
- Create the Gaussian kernel.
- Apply the Gaussian kernel to the image.
- Save the blurred image.

☐ **Pseudocode**

Function create_gaussian_kernel(kernel_size, sigma):

Generate a linear space from -(kernel_size - 1) / 2 to (kernel_size - 1) / 2

Create a meshgrid from the linear space

Compute the Gaussian function on the meshgrid

Normalize the kernel so that its sum is 1

Return the kerne

Function apply_kernel(image, kernel):

Determine the radius of the kernel

Normalize the image to the range [0, 1]

Pad the image using reflection padding

Create an empty image for the blurred result

For each pixel in the image (excluding padding):

For each color channel:

Compute the sum of the product of the kernel and the surrounding pixels

Assign the result to the corresponding pixel in the blurred imag

Clip the values of the blurred image to the range [0, 255]

Convert the image to 8-bit format

Return the blurred image

Function gaussian_blur(kernel_size, sigma, image_path):

Load the image and convert it to a NumPy array

Ensure the kernel size is odd

Create the Gaussian kernel

Apply the Gaussian kernel to the image

Save the blurred image

Return the path to the blurred image

☐ **Block Diagram**

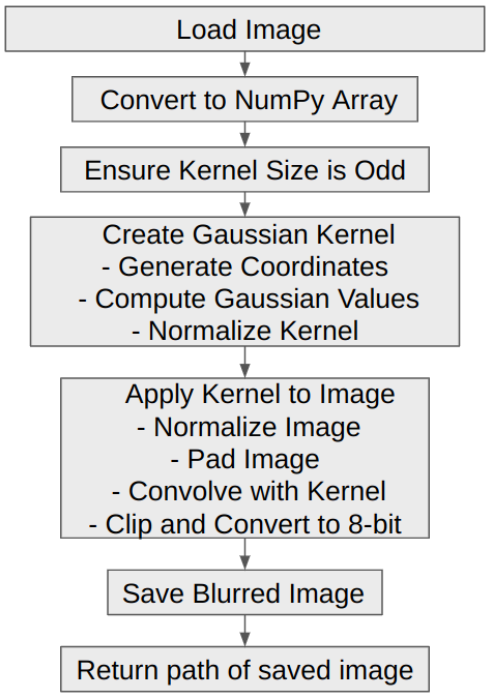

Fig-24: Block Diagram of Gaussian Blurring Tool

☐ **UI**

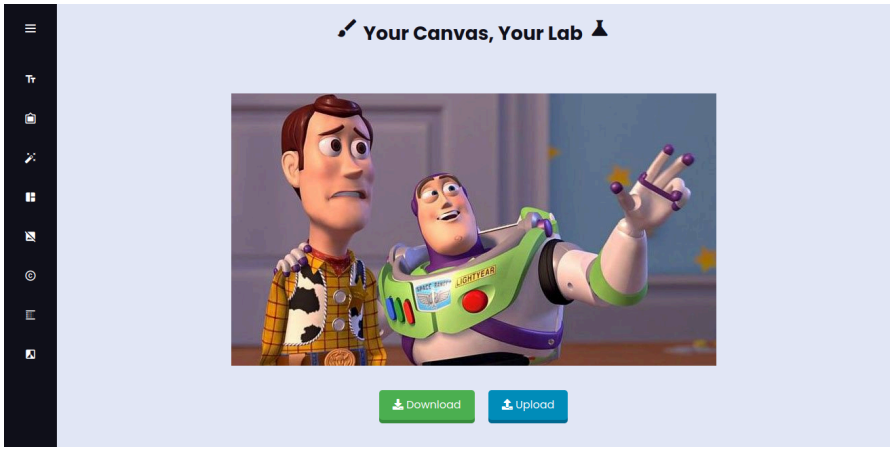

Fig-25: Input Image

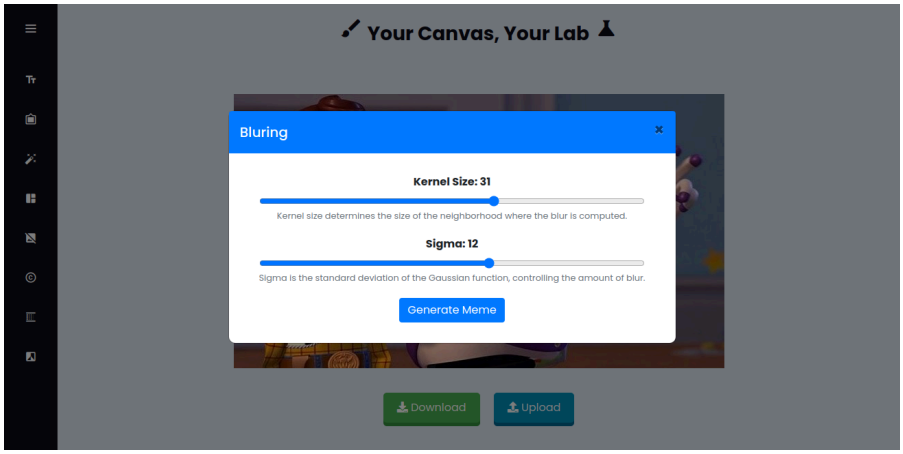

Fig-25: Adjusting Parameters

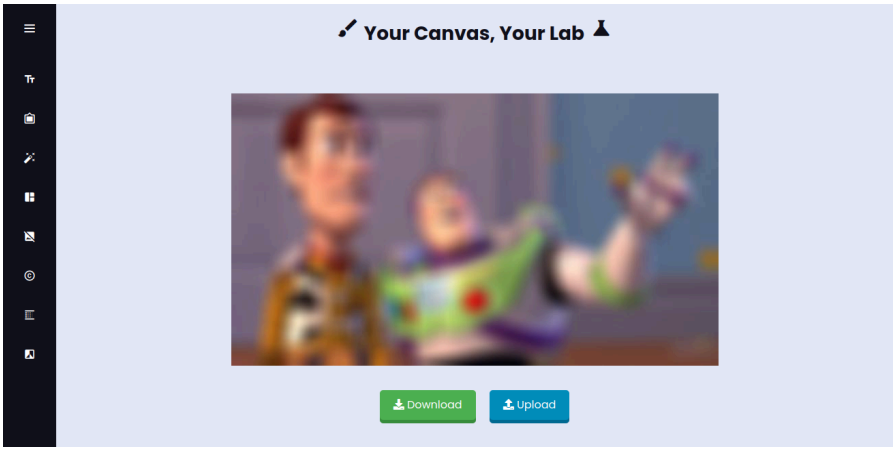

Fig-26: Output Image

Black-White

- ☐ **Algorithm**

Define the function `convert_to_black_and_white` which converts an image to grayscale:

- Open the image using PIL.
- Convert the image to grayscale.
- Save the processed image and return its path

☐ **Pseudocode**

function convert_to_black_and_white(image_path):

open image with PIL

convert the image to grayscale

save the grayscale image

return path of saved image

☐ **Block Diagram**

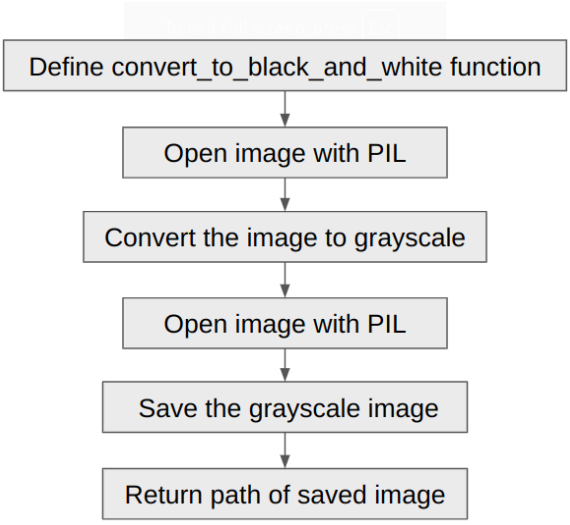

Fig-27: Block Diagram of Black-White Tool

☐ **UI**

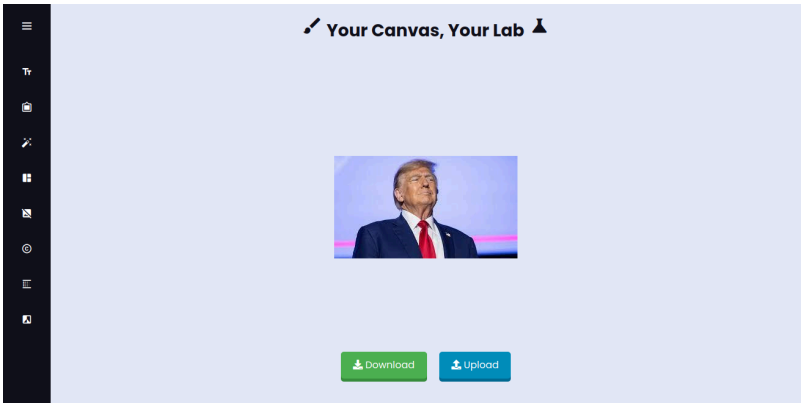

Fig-28: Input Image

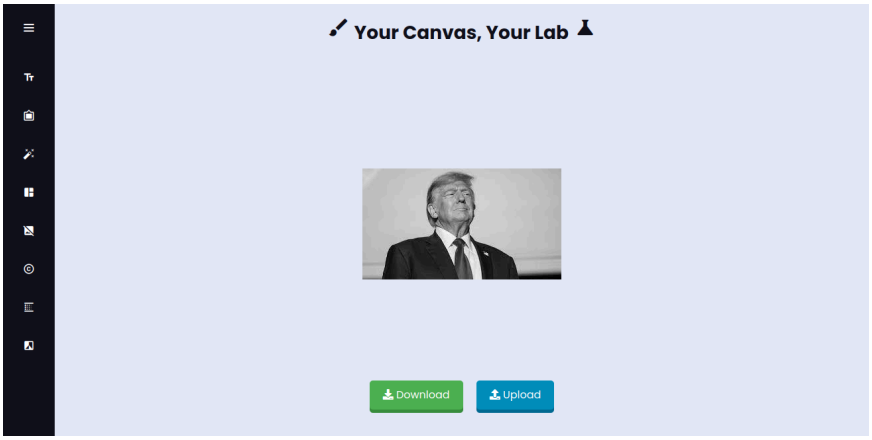

Fig-29: Output Image

Application:

- Social Media Engagement: Enhancing posts with custom memes.
- Educational Tool: Demonstrating image processing and computer vision.
- Marketing and Branding: Engaging audiences with viral content.
- Personalized Content Creation: Customizing memes for individual expression.
- Research and Development: Experimenting with image processing algorithms

Discussion:

The meme-generator project integrates a variety of image processing and computer vision techniques, enabling users to manipulate images with features like text addition, collage creation, histogram equalization, and background removal. Implementation challenges included ensuring compatibility across different image formats, optimizing performance for real-time interaction, and refining user interface design for intuitive use. User feedback emphasized the need for improved text handling and faster processing times, which were addressed through iterative updates and algorithm optimizations. Overall, the project successfully combines practical application with educational value, serving as a versatile tool for both casual users and academic settings.

Conclusion:

In conclusion, the meme-generator project has achieved its goals of providing a user-friendly platform for creating and sharing memes while exploring advanced image processing techniques. With its diverse applications across social media, education, marketing, and research, the project demonstrates the intersection of technology and creativity. Future enhancements could focus on expanding the range of editing tools, enhancing real-time collaboration features, and integrating cloud-based storage for seamless user experience and scalability. Through this project, valuable insights were gained into the complexities of image manipulation and the potential for further innovation in digital content creation.