On Demand Image Rescaling (ODIR)

By

Rauf Ali

M. Umer Javed

Usama Bin Akhtar

Malik Muneeb Ur Rehman

Supervised by:

Dr Attiq Ahmad

Submitted to the faculty of Department of Electrical Engineering,
Military College of Signals, National University of Sciences and Technology, Islamabad,
in partial fulfillment for the requirements of B.E Degree in Electrical Engineering.

June 2024

In the name of ALLAH, the Most benevolent, the Most Courteous

CERTIFICATE OF CORRECTNESS AND APPROVAL

This is to officially state that the thesis work contained in this report

“On Demand Image Rescaling”

is carried out by

Rauf Ali

M. Umer Javed

Usama Bin Akhtar

Malik Muneeb Ur Rehman

under my supervision and that in my judgement, it is fully ample, in scope and excellence, for the degree of Bachelor of Electrical Engineering in Military College of Signals, National University of Sciences and Technology (NUST), Islamabad.

Approved by

Supervisor

Dr Attiq Ahmad

Department of EE, MCS

Date: ___________

DECLARATION OF ORIGINALITY

We hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

ACKNOWLEDGEMENTS

Allah Subhan’Wa’Tala is the sole guidance in all domains.

Our parents, colleagues, and most of all the supervisor, Dr Attiq Ahmad without your guidance.

The group members, who through all adversities worked steadfastly.

Plagiarism Certificate (Turnitin Report)

This thesis has a 10% similarity index Turnitin report endorsed by Supervisor is attached.

Malik Muneeb Ur Rehman

NUST Serial no 00000343838

Rauf Ali

NUST Serial no 00000359578

M. Umer Javed

NUST Serial no 00000337533

Usama Bin Akhtar

NUST Serial no 00000331869

Signature of Supervisor

ABSTRACT

This project presents an HTTP-based image transformation service. The service supports various transformation operations, such as resizing, cropping, rotating, and filtering. The goal of the project is to develop and implement a RESTful API for the corresponding service that will enable users to request transformations of images by sending HTTP requests. One can view the project as a kind of a set of tools that help to process images online. The implementation of the project is in Python programming language and makes use of a popular micro web framework called Flask. Users who have registered for the service receive their unique API keys that they use to authenticate themselves and gain access to the API endpoints. The project allows users to upload files and indicate URL to the image for it to be processed. Apart from resizing and rotation, filtering options allow users to change an image the way they want in terms of processing. Besides, the project has ensured that it is easy to integrate with other applications by providing SDKs and complete documentation. Therefore, On Demand Image Rescaling is highly available and flexible software for image processing tasks.

Table of Contents

List of Figures

Chapter 1: Introduction

Nowadays, when visual content plays a vital role in attracting users in the digital world, the popularity of image manipulation and transformation services is rapidly growing. One of the leading companies in offering such services is Cloudinary, which provides dynamic URL transformations allowing the easy adaptation of images to a web or mobile application's new look and feel. Nevertheless, although such services effectively and conveniently solve organizations' problems, companies frequently opt for building in-house solutions instead of buying them from third-party vendors. The main reasons include price, and the lack of possibility to control the transformation process.

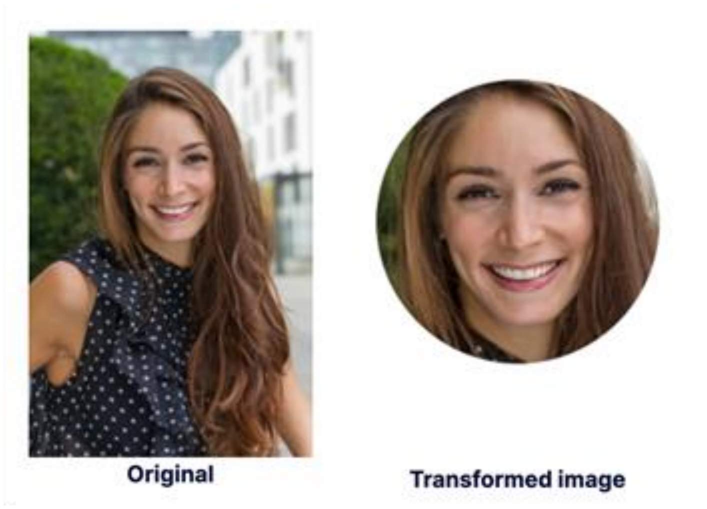


Figure 1: An example from Cloudinary.

On-demand image rescaling provides adjustment in the size of images based on end devices or user interface. These images are manipulated (resized, cropped, or flipped) as needed, making them viable for different screen sizes. This system's implementation on websites can enhance performance, improve user experience, and load faster by giving appropriately sized images to users. It may also involve smart detection of objects, references, and important points in images (faces) to ensure quality and size of original image is preserved. Additionally, methods like using Content Delivery Networks (CDNs) can also be used to achieve this system. Many websites and online portals are available for this specific purpose [1][2].

By looking at modern technology and increasing demand of visuals mainly images in digital world, this project aims develop an HTTP-based image transformation service. The main goal is to create a strong and explorable system capable of managing different images and providing transformation operations, ranging from basic resizing, cropping to advanced filtering, and rotating functionalities. Design and implementation of a RESTful API (Application Programming Interface) that allows users to request image transformations via standard HTTP requests is the central part of developing the system.

By developing this service, business entities, software houses and many organizations will gain the advantage to manage and customize their image transformation processes according to their preferences. This project not only addresses the need for in-house image transformation solutions but also opens opportunities for optimization in terms of digital content. This project may also be helpful in marketing agencies where it empowers organizations to elevate their visual experiences and drive more engagement with their target audience.

1.1 Overview

In the modern world of digitalization, the growing tech field and development in the fields of AI, Machine Learning and Image processing have become the most influential developments in the daily life of mankind. Every other person now has a social account or is present on various online platforms. According to one survey As of October 2023, the number of people using social media is over 4.95 billion worldwide, with the average user accessing 6.7 social media platforms monthly.

Similarly, many business owners have their own website, applications for User manual that require many back-end operations of image transformation. Most of the time big companies like Facebook (Meta), Instagram have built in image resizer algorithms, you must have noticed a big resolution image (1920x1080) is automatically downsized if uploaded as a profile picture. But some companies buy these services form third-party applications like cloudinary that's where the need for project raised.

As the name suggests, an on-demand image rescaling project involves developing a system that can dynamically resize images based on specific requirements or user requests. The project involves technology stack (coding languages like Python, Django), image rescaling algorithms, back-end development, image storage (cloud), front-end development, authentication and authorization and quality control. These APIs will allow companies to implement and manipulate their private data (images) as per their choice and save their time.

It is the need of the hour to enable users ease and control over their images on demand. Hence in our proposed system of image rescaling, we not just focus on the image operations but also encounter cost consideration to ensure user experience.

1.2 Problem Statement

There is a growing need for image manipulation and transformation services on-demand. Cloudinary is one such service that provides dynamic URL transformations for adjusting images to fit the graphic design of an application. However, organizations may prefer to have their own in-house image transformation service for various reasons, such as data privacy or cost considerations. The points that need to be addressed are as follows,

1. Current solutions often require manual intervention to resize and optimize images for various devices, which is time-consuming and prone to errors.
2. Uploading high-resolution images to serve all device types results in excessive bandwidth consumption, leading to slower loading times and increased data costs for users.
3. Images that are not properly optimized for different devices can lead to distorted or pixelated visuals, compromising the overall user experience and potentially deterring user engagement.
4. Websites and applications with user-generated content require a flexible and scalable solution to dynamically resize images based on user preferences and device characteristics.

1.3 Proposed Solution

The aim is to develop an On-Demand Image Rescaling System to address the challenges linked with managing, optimizing, and manipulating images across various digital platforms. This system provides dynamically resizing and optimizing images in real-time, ensuring an optimal user experience while considering factors like storage constraints, and manual intervention into account.

The main part or crux of this project is to develop an HTTP-based image transformation service. The service should support various transformation operations, such as resizing, cropping, rotating, and filtering. The project will involve designing and implementing a RESTful API for the service, which will allow users to request image transformations using HTTP requests.

By establishing an in-house image transformation service, organizations will gain greater control over their digital assets while dynamic resizing algorithm, API-driven architecture, automated image processing, user-generated content support will ensure users cost and quality experience.

1.4 Working Principle

The project mainly works on the principles of image processing with back-end development. The project is divided into different parts and each part is inter-connected with the next one. The list of these parts is as under:

- Image Upload.
- Image Transformation Operations.
- Dynamic Resizing.
- API Driven Access.
- On Demand Processing.
- User-Generated Content Support.
- Monitoring and Maintenance.

1.4.1 Image Upload:

The process begins when a user uploads an image through the system's interface, typically via a website, mobile app, or API call. The original image is securely transmitted to the system's backend for processing and storage.

1.4.2 Image Transformation Operations:

Upon receiving an image upload request, the system triggers an image processing pipeline responsible for resizing and optimizing the image.

The pipeline may consist of multiple stages, including image resizing, format conversion, quality optimization, and metadata extraction. The image processing or operations our project provides are cropping, rotating, resizing, filtering, and morphological operations [3].

- **Crop Image:**

  Cropping is a technique used in image processing to remove unwanted parts of an image and focus on a specific area or subject. It involves selecting a portion of the original image and discarding the rest. This process allows users to adjust the composition of the image, remove distractions, or highlight specific details.

  For example, in the picture below one wants to crop the face of the lady in the image the required algorithm would crop the face.

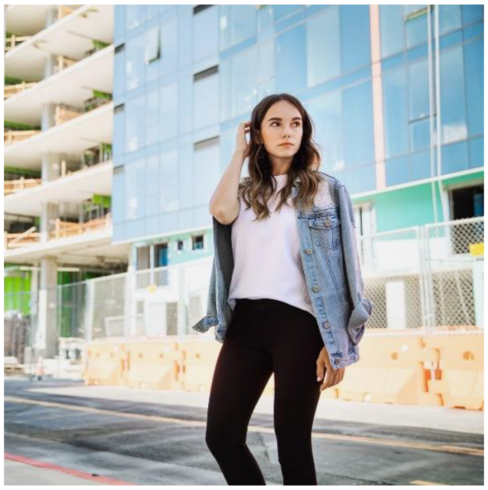


Figure 2-a: Original Image

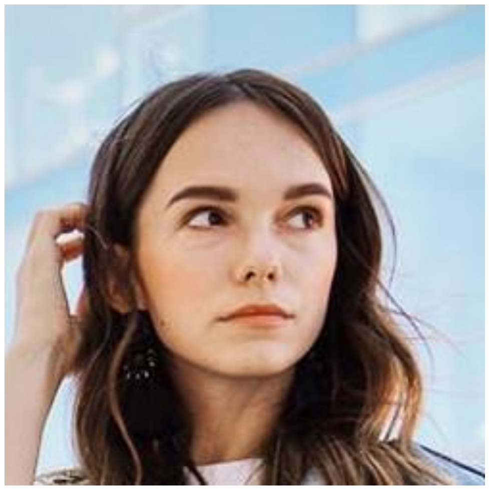
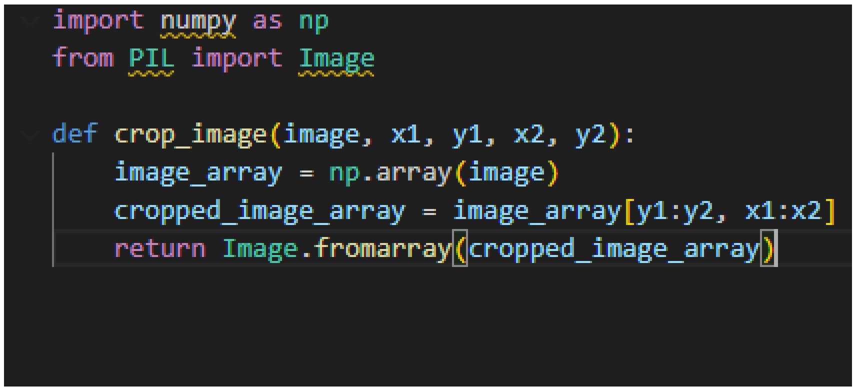


Figure 2-b: Cropped Face



```
import numpy as np
from PIL import Image

def crop_image(image, x1, y1, x2, y2):
    image_array = np.array(image)
    cropped_image_array = image_array[y1:y2, x1:x2]
    return Image.fromarray(cropped_image_array)
```

Figure 3: Code for Cropping

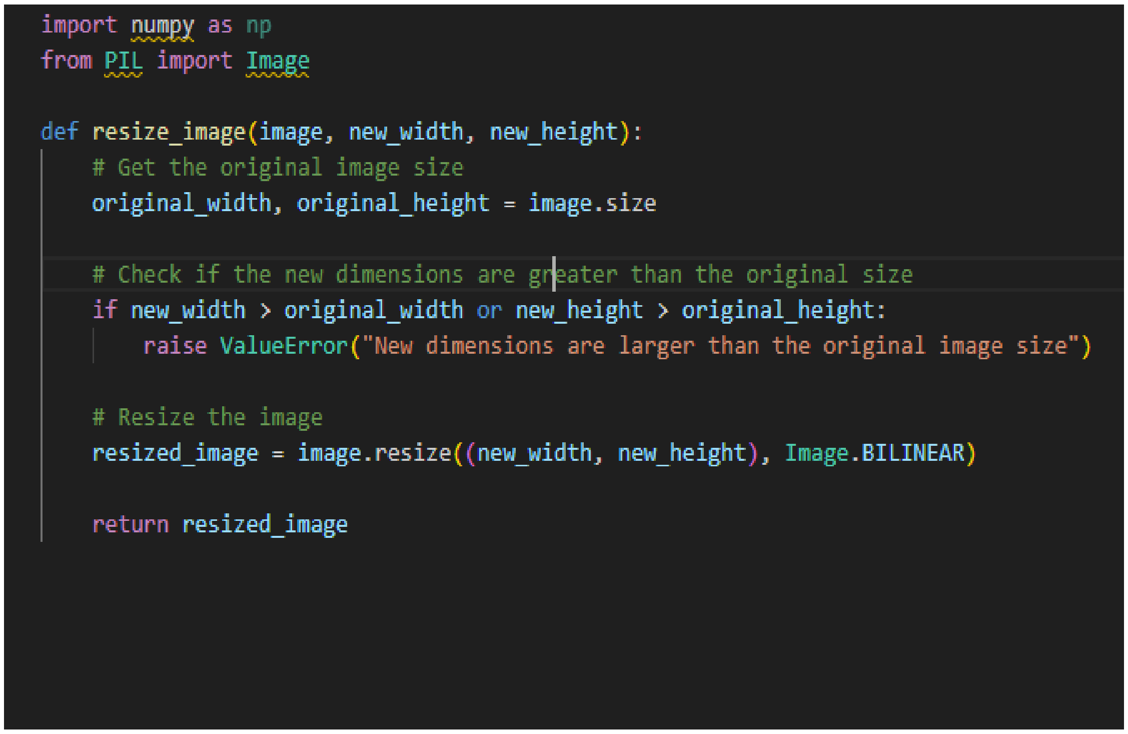
You can specify a region of the original image to crop by giving the ‘**x1**’ and ‘**y1**’ coordinates as the top left corner of the region together with ‘**x2**’ and ‘**y2**’ coordinates as the bottom right corner of the region.

- **Resize Image:**

Resizing in the context of digital images refers to the process of changing the dimensions (width x height) of an image. This can involve making the image larger (upsizing) or smaller (downsizing) while maintaining its aspect ratio (e.g. 4:3 or 16:9) or altering it. Resizing is a common operation in image processing and is performed for various reasons, including display, file size, printing.

A function **resize image** that resizes an image using bilinear interpolation, , in which we use the four nearest neighbors to estimate the intensity at a given location would be defined. It takes an input image and desired new width and height as parameters. Only downscaling is done for now.

$$v(x, y) = ax + by + cxy + d$$



```python
import numpy as np
from PIL import Image

def resize_image(image, new_width, new_height):
    # Get the original image size
    original_width, original_height = image.size

    # Check if the new dimensions are greater than the original size
    if new_width > original_width or new_height > original_height:
        raise ValueError("New dimensions are larger than the original image size")

    # Resize the image
    resized_image = image.resize((new_width, new_height), Image.BILINEAR)

    return resized_image
```

Figure 4: Code for Resizing

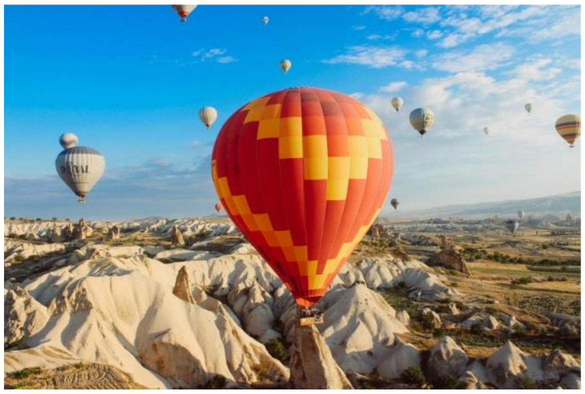


Figure 5-a: Original Image (1000 x 667)

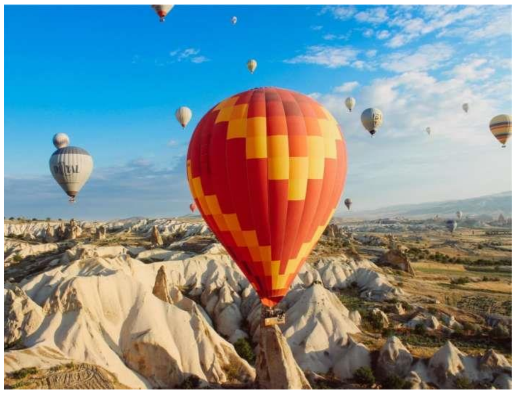


Figure 5-b: Resized (700 x 530)

The function checks if the new dimensions are smaller than or equal to the original image size. If they are, it resizes the image using bilinear interpolation and returns the resized image. If the new dimensions exceed the original size, it gives a value error.

- **Rotate Image:**

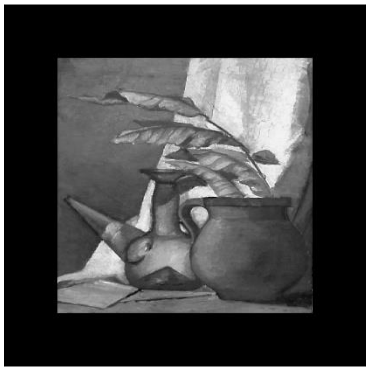


Figure 6-a: Original Image

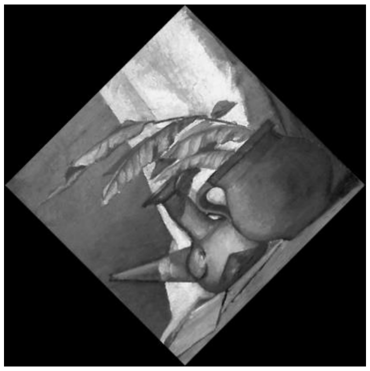


Figure 6-b: Rotated 45°

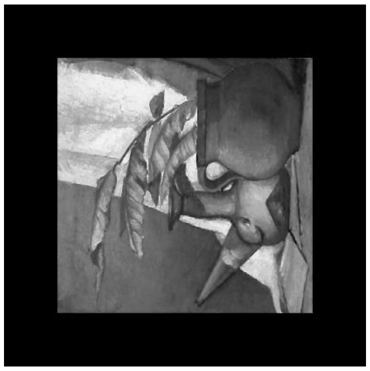


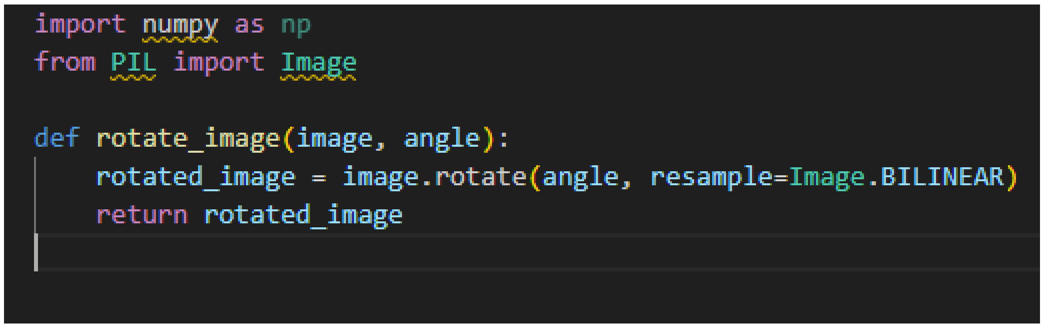
Figure 6-c: Rotated 90°

Rotating an image means changing its orientation by an angle, mostly in degrees. This process edits the arrangement of pixels in the original image to create a new visual representation.

When rotating an image, the pixels of original image need to be transformed to new positions according to angle of rotation.

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Bilinear interpolation is the most common method used to estimate the intensity values of pixels at coordinates by interpolating between the closest pixels.



```python
import numpy as np
from PIL import Image

def rotate_image(image, angle):
    rotated_image = image.rotate(angle, resample=Image.BILINEAR)
    return rotated_image
```

Figure 7: Code for Rotating

Below is how rotating an image workflow looks like,

Select Rotation Center

The common center of rotation is the center of the image.

Calculate New Pixel Positions

Determine the new position that the input pixel element should map to if the rotation is carried out upon the input image with respect to the rotation angle. Do an interpolation based on the closest four-pixel positions to determine the new pixel's intensity value.

- **Gaussian Filter:**

A Gaussian filter is a type of smoothing (lowpass) spatial filter used to reduce sharp transitions in intensity. Convolving a smoothing kernel with an image blurs the image, with the degree of blurring being determined by the size of the kernel and the values of its coefficients.

To develop backend code, define functions to blur an image using a Gaussian filter. First create a Gaussian kernel based on specified parameters such as size and sigma.

$$K(x, y) = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{\left(x-\frac{N}{2}\right)^2+\left(y-\frac{N}{2}\right)^2}{2\sigma^2}\right)}$$

N (denoted as **size** in the code) is the size of the kernel (assuming a square kernel for simplicity).

x and y are the coordinates within the kernel, ranging from 0 to $N-1$.

$\frac{1}{2\pi\sigma^2}$ is the normalization factor ensuring that the total sum of the Gaussian kernel is 1.

exp $(-\frac{(x-\frac{N}{2})^2+(y-\frac{N}{2})^2}{2\sigma^2})$ is the exponential function that defines the Gaussian distribution centered at the middle of the kernel $(\frac{N}{2},\frac{N}{2})$.

Then, apply this kernel as a filter to the input image, effectively blurring it to reduce noise or smooth out details. Finally, it returns the processed image. The code is given later in the fourth chapter.

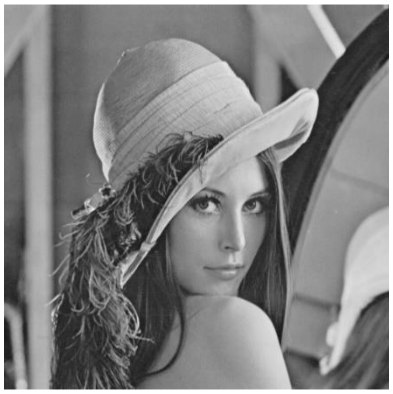


Figure 8-a: Original Image of a girl

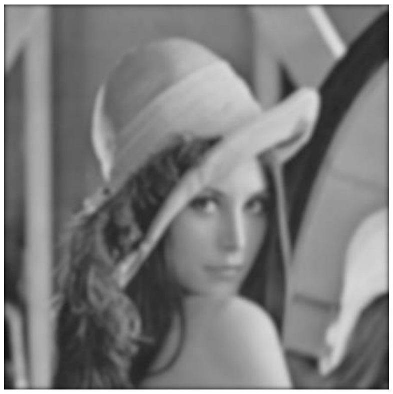


Figure 8-b: Gaussian-var-400

- **Apply Sharpening Filter:**

Sharpening, in the context of image processing, refers to a technique used to enhance the edges and details in an image. It works by accentuating the high-frequency components of the image, which typically correspond to edges and transitions between different regions of the image. The code is given later in the fourth chapter.

It works by computing the second derivative of the image, highlighting areas where the intensity changes abruptly.

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

The above equation represents Laplacian kernel derived through differentiation.

The kernels used in the function (like Unsharp Masking and Laplacian) highlight these differences:

Unsharp Masking Kernel: This kernel enhances edges by subtracting a blurred (average) version of the image from the original image. It’s called "unsharp" because it effectively subtracts the unsharp (blurred) version to enhance the sharpness.

$$kernel = np.array\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Laplacian Kernel: This kernel emphasizes regions of rapid intensity change, which corresponds to edges in the image.

$$kernel = np.array\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

By convolving the image with these kernels, the function enhances edges and fine details, resulting in a sharpened image.

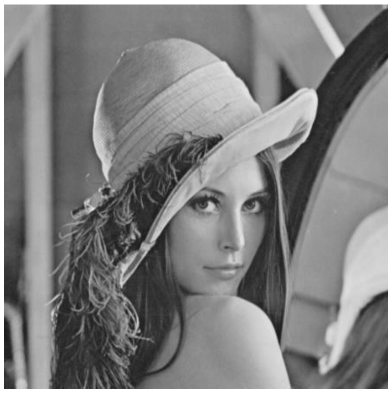


Figure 9-a: Original Image

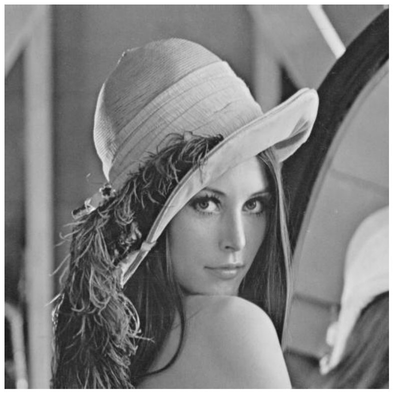


Figure 9-b: Sharpened Image

Process involves enhancing the contrast between neighboring pixels, particularly along edges. This is typically achieved by increasing the intensity difference between adjacent pixels that form an edge, making the transition between light and dark regions more pronounced.

- **Apply Morphological Operations**:

  Morphological operations are a set of image processing techniques used to analyze and manipulate the structure of features within an image. These operations are primarily used for preprocessing tasks such as noise removal, feature extraction, and object segmentation. The code is given later in the fourth chapter.

  Here's a brief description of each operation:

Erosion: Erosion operation opens the white regions in the image and makes the black regions larger. It can eliminate small white noise, disjoint objects, and reduce the overall size of foreground objects.

These operations are represented by structuring elements and a set A of foreground pixels. With A and B as sets in Z^2 the erosion of A by B denoted by A $\ominus$ B is given by,

$$\mathrm{A} \ominus \mathrm{B} = \{ z \mid (B)_z \subseteq \mathrm{A} \}$$

In words, this equation indicates that the erosion of A by B is the set of all points z such that B, translated by z, is contained in A.

Dilation: Dilation operation opens the white regions in the image and makes the black regions smaller.

$$\mathrm{A} \oplus \mathrm{B} = \left\{ z \mid [(\hat{B})_z \cap \mathrm{A}] \neq \emptyset \right\}$$

This equation is based on reflecting B about its origin and translating the reflection by z, as in erosion. The dilation of A by B then is the set of all displacements, z, such that the foreground elements of Bˆ overlap at least one element of A

Opening: An opening is just a variation. First, the erosion is applied then, dilation is applied using the same kernel. The opening of set A by structuring element B, denoted by A is defined as,

$$\mathrm{A} \circ \mathrm{B} = (\mathrm{A} \ominus \mathrm{B}) \oplus \mathrm{B}$$

1.4.3 Dynamic Resizing:

The uploaded image is dynamically resized by the system. The size of the image is based on the type of device, the screen resolution, and the user's preferences. Image resizing algorithms, including bicubic interpolation and approaches based on deep learning, change the dimensions of the image while maintaining visual quality.

1.4.6 API Driven Access:

The system Provide developers with RESTful APIs to query for images that are resized in the background. Developers can adjust the image dimensions, set quality, and change the optimization settings by using the API. This feature allows easy integration with websites, mobile applications, and many other digital platforms.

1.4.7 On Demand Processing:

Resizing and optimizing workloads are done only on-demand, upon user request, or in an automated workflow. Serverless computing technologies such as AWS Lambda or Google Cloud Functions can be used to run processing workloads efficiently and affordably.

1.4.8 User Generated Content Support:

The system is designed to support user-generated content. Users can upload images in many different formats and resolutions. The images that are uploaded are resized and optimized in real-time to ensure acuity and consistency across various gadgets and operating systems.

1.4.9 Monitoring and Maintenance:

A monitoring and maintenance mechanism is incorporated into the system to ensure operational availability and performance. Monitors are used to gather all major metrics from the system and alert on error or deviations.

1.5 Objectives

1.5.1 General Objectives:

“To develop an in-house back-end software integrated with front-end prototype powered by image processing techniques, providing a smart API driven structure that would reduce users coding time and increase quality, experience and privacy.”

1.5.2 Academic Objectives:

- Designing the API (Application Programming Interface).
- Implementing the API.
- Implement the core logic for various transformation operations, such as resizing, cropping, rotating, and filtering.
- Implement secure authentication mechanisms to control access to the image transformation service.
- Testing and Validation.
- Documentation.
- Deployment and Scalability. (Conditional)
- Optional Enhancements. (Conditional)

1.6 Scope

The project scope and aim is to create an image transformation service, predominantly based on HTTP. The project should support multiple transformation operations including changing the **size**, **cropping** several parts of it, **rotation**, and some **filters**. Particularly, it is possible to develop a RESTful API for such a project which provides the ability to perform all the mentioned-above operations on the image that users could send by the HTTP protocol.

1.7 Deliverables

1.7.1 Image Transformation Operations:

The software may provide image transformation operations like resizing, cropping, filtering, and morphological operations.

1.7.2 Developed APIs:

APIs for various image operations such as cropping, rotating, resizing, and filtering may be developed. End points and request/response structure (GET, POST) for each operation should be defined. It should also be able to handle various image formats (JPEG, PNG etc.) as inputs.

1.7.3 Integration and Testing:

To ensure digital ease testing the code is the main thing. The integration of the front-end prototype with the back-end hard code may be established. Errors review and minimize.

1.8 Relevant Sustainable Development Goals

Two main Socio-Economic Issues that our Project Addresses,

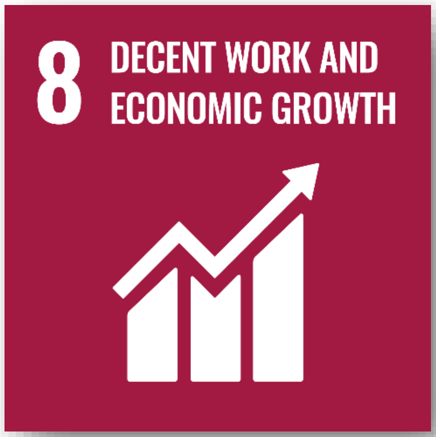


Figure 10: SDG 8

Figure 11: SDG 9

1.8.1 Decent Work and Economic Growth (SDG 8):

As businesses and industries adopt digital technologies for image processing and multimedia applications, there is a growing demand for skilled workers proficient in areas such as software development, data science, and digital content creation. Hence increasing job creation and skill development.

The image processing API democratizes access to advanced image processing capabilities by providing a scalable and accessible platform for developers and businesses of all sizes.

1.8.2 Industry, Innovation, and Infrastructure (SDG 9):

The project promotes industrialization by facilitating the integration of image processing technologies into various industries, such as media and entertainment, e-commerce, healthcare, and manufacturing.

By providing a scalable and reliable platform for image processing tasks, the API contributes to the development of digital infrastructure, which is essential for supporting the growth of digital economies and fostering innovation.

1.9 Structure of Thesis

Chapter 2 contains the literature review and the background and analysis study this thesis is based upon.

Chapter 3 contains the design and development of the project.

Chapter 4 introduces detailed evaluation and analysis of the code.

Chapter 5 contains the conclusion of the project.

Chapter 6 highlights the future work needed to be done for the commercialization of this project.

Chapter 2: Literature Review

In today's digital age, the demand for multimedia content creation and manipulation has surged across various industries, including media and entertainment, e-commerce, healthcare, and education. Image processing APIs are now critical for developers and firms looking to offer high-quality digital content creation and productivity solutions. The API offers a uniform set of commands for implementing image modifications that range from fundamental changes, including cropping and resizing, to far more complex methods such as filtering and morphological modifications. The following are the concepts that will be covered throughout the review: the role of image processing APIs in enabling reliable productivity and digital content creation tool arrest, related work, and its application to the current literature.

- Historical Background
- Traditional Method for Manipulating Images
- Existing solutions and their drawbacks
- Research Papers

2.1 Industrial background

Scope of the project and most important priority bodies to provide service for the image transformation service parts interfaces. History milestones, though down scaling, and upscaling enhanced our service it serves future needs,

- In the 1980s and 1990s, advances in digital image processing were made rapidly, buoyed by breakthroughs in the development of computing software and hardware.
- During his time, researchers and academicians have loomed as big as to offer fundamental building blocks and algorithms for image analysis, shape theory, and dynamic vision.

- Image processing commercial software products, including Photoshop, have surfaced and given end users strong tools that manipulate digitalized pictures.
- In the early 2000s, image file formats and protocols were standardized, including JPEG, PNG, and TIFF, which made it easier to work with imaging systems and enabled data exchange between different software applications.
- Furthermore, web technologies and cloud computing also thrived in the late 2000s and 2010s, resulting in a rapidly increasing need for accessible and scalable image processing solutions.
- Consequently, image processing APIs and libraries were launched, providing developers with a programmable interface for a variety of image manipulation functions.
- Open-source projects and cloud APIs like OpenCV, Pillow, Google Cloud Vision, and Amazon Recognition became increasingly popular solutions for image processing.

2.2 Traditional Methods for Manipulating Images:

Before the development of image processing APIs became standardized, traditional methods consisting of manual image processing, specialized software, and building custom algorithms were used for similar purposes. These included the following:

- **Manual Editing**

  First, even just a few decades ago, many tasks that could be easily performed automatically with modern-day automated image processing protocols, had to be done manually. The main protocols for image creation were graphic design software such as Adobe Photoshop [4].

  Given that such software supported only manual protocols of image editing, graphic designers and digital artists had to manually "recreate" images by using a variety of tools such as selection tools, brushes, and filters.

- **Custom Scripts and Algorithms**

  In other cases, when such tasks required complex image transformation that could not be computerized yet, software developers and engineers had to write custom scripts and algorithms to process and analyze images.

  These custom solutions require expertise in programming and image processing techniques and are applicable in the field through trial and error.

- **Desktop Software**

  Several desktop software applications, such as Adobe Photoshop, GIMP, CorelDRAW, and others, equipped users with an extensive spectrum of tools and options to edit and manipulate images easily. On the one hand, they allowed users to perform such routine tasks as cropping, scaling, color correction, texturing, filtering, etc. On the other hand, most of them had a massive array of functions and were complex for non-experienced users.

- **Command-Line Tools**

  Command-line batch processing programs, e.g., ImageMagick [5] and GraphicsMagick, unlocked the potential for developers to automate the manipulation of images. In other terms, they empowered programming scripts through integration, ensuring the speedy, automated solution of processing tasks.

- **Library-Based Solutions**

  Third, programming libraries and frameworks, such as OpenCV [6] and MATLAB's Image Processing Toolbox [7], widened developers' opportunities. They afforded the creation of original solutions for image filtering processes based on the existing algorithms.

- **Specialized Hardware**

  Also, hardware devices, such as digital imaging scanners and printers, were used for capturing and printing digital images.

  These devices have built-in software's that allow users to perform basic image manipulation tasks, such as scanning, resizing, and auto color adjustment.

This traditional method required a lot of training, expertise, and skill to apply practically on images. The invention of image processing APIs has allowed access to image manipulation tasks, making it further easier for developers and businesses to incorporate advanced image processing functionality into their applications with minimal overhead.

2.2.1 Why opt for Image Editing APIs?

As such, image editing APIs and traditional image editing software serve different purposes and players with varying needs and expertise. Image editing APIs are often the way to go for businesses seeking to edit many images flawlessly and with limited duration, such as those in e-commerce as highlighted. Image editing done through a traditional or manual process is ideal for in-depth players that prioritize precision and creative freedom and are willing to put in the time and effort. However, APIs like Photo room's API have it all since they offer AI photo editor services eliminating the background for easy positioning and unwanted objects as well as generating reflections, shadows, and new backgrounds. Most importantly, these APIs can be easily integrated into a business's image processing workflow, allowing businesses to immediately benefit from cutting-edge photo editing technologies.

On the other hand, traditional image editing software is expensive and eats time, skill, and training to use. While traditional editing techniques offer more quality, they can be energy draining especially when dealing with bulk amount of work.

2.2.2 Our Approach

The project's purpose is to create a highly effective and user-designed image processing API that will allow for standard image changes and secure user authentication for personal privacy, confidently, and cost.

2.3 Existing Solution and their drawbacks:

There are many solutions available in the market already with some benefits and disadvantages. Some of them are,

- Traditional Software Applications (e.g., Adobe Photoshop, GIMP)
- Cloud-based Image Processing APIs (e.g., Google Cloud Vision, Cloudinary):

  These existing solutions provide a range of image processing functionalities and vary in terms of pricing, ease of use, scalability, and integration options. Depending on the specific requirements and constraints of an application, developers may choose one of these existing solutions or opt for building a custom image processing system.

2.3.1 Traditional Software Applications:

Some of the drawbacks of traditional software that provide the services are,

- *Steeper learning curve*: Traditional software applications often have complex user interfaces and require specialized knowledge to use effectively.

- *Limited scalability*: Processing large volumes of images or implementing batch operations can be time-consuming and resource intensive.

- *High upfront costs*: Purchasing licenses for commercial software applications can be expensive, particularly for small businesses or individual users.

2.3.2 Cloud-based Image Processing APIs:

These are better in terms of facilities and recommended far more than the other two but still some things like cost security and bonding remain issues. Some examples include Google Cloud Vision, Cloudinary etc. Few points that need to be addressed are,

- *Vendor lock-in*: Depending on a specific cloud provider's API may result in vendor lock-in, making it difficult to switch providers or migrate to a different platform.
- *Cost considerations*: Cloud-based APIs typically charge based on usage, which can lead to unpredictable costs for applications with fluctuating workloads or high volumes of image processing tasks.
- *Privacy and security concerns*: Sending sensitive image data to third-party cloud services raises privacy and security concerns, particularly for industries with strict data protection regulations.

2.4 Research Papers:

Although most of our work is based on image operations like cropping, rotating, resizing, and filtering that require digital image processing knowledge, the field of image editing is vast. AI based image editing detection and other scenarios have been studied thoroughly through these papers. These research papers provide valuable insights into the state-of-the-art image processing APIs, their capabilities, limitations, and potential areas for improvement. Researchers and practitioners in the field of computer vision, remote sensing, and cloud computing can benefit from the findings and recommendations presented in these papers. Here is a list of them,

- "Research on Digital Image Processing Technology and Its Application" by Zhou et al. (2018) [8].

  This paper depicts the tremendous advances made in digital image processing technology and its wide application. It highlights the importance of more research into unifying artificial intelligence and enabling efficient logical structures for better processing results.

The future development of digital image processing should be three words: miniaturization, intelligence, and convenience.

- “Cloud-based image processing services: State-of-the-art and challenges”. *Published in* 2020 7th International Conference on Computing for Sustainable Global Development (INDIACom).[9]

  This paper comes up with the conclusion that cloud based image processing services are high scalable and cost-effective Cloud image processing has great potential competitiveness compared to traditional opponent like conventional mini cloud & pic hosting environments.
- "Comparative Analysis of Image Processing Techniques Using Cloud Computing Paradigm". *Published in* 2017 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT).[10]

  It focuses and empathizes on using cloud computing for image processing, its comparative analysis, advantages and future directions for the development of software like these.

These research papers provide current state-of-the-art in cloud-based image processing services, including architectures, techniques, applications, and challenges.

Chapter 3: Software Requirements and Specifications

3.1 Introduction:

This chapter gives a full description of the On Demand Image Rescaling System. It will explain the purpose, features, interfaces, functionality, entire process, constraints, and the application's reaction to external stimuli. It is intended for stakeholders and system developers.

An image processing API that performs various operations such as cropping, resizing, rotating, and applying filters to images. Importance of defining software requirements and specifications to ensure clarity, quality, and successful implementation.

3.2 Functional Requirements:

3.2.1 User Authentication:

- **Registration Endpoint**: Allows new users to register and receive an API key.

  Input: Username and password.

  Output: API key.

- **API Key Authentication**: Each image processing request must include a valid API key in the header.

  Input: API key.

  Output: Authorization status.

3.2.2 Image Operations:

- **Crop Image**:

  Endpoint: '/crop'.

  Inputs: Image file or URL, coordinates (x1, y1, x2, y2).

Outputs: Cropped image.

- **Resize Image**:

  Endpoint: ‘/resize’.

  Inputs: Image file or URL, new width, new height.

  Outputs: Resized image.

- **Rotate Image**:

  Endpoint: ‘/rotate’.

  Inputs: Image file or URL, rotation angle.

  Outputs: Rotated image.

- **Apply Gaussian Filter**:

  Endpoint: ‘/gaussian_filter’.

  Inputs: Image file or URL, kernel size, sigma.

  Outputs: Filtered image.

- **Apply Sharpening Filter**:

  Endpoint: ‘/sharpening_filter’.

  Inputs: Image file or URL, filter name.

  Outputs: Sharpened image.

- **Apply Morphological Operations**:

  Endpoint: ‘/morphological_operation’.

  Inputs: Image file or URL, kernel name.

  Outputs: Processed image.

3.3 Non-Functional Requirements:

3.3.1 Performance:

- The system should handle multiple simultaneous requests efficiently.
- The image processing operations should be performed with minimal latency.

3.3.2 Security:

- User passwords must be stored securely in the database.
- API keys must be unique and securely generated.
- Ensure secure communication (e.g., using HTTPS).

3.3.3 Usability:

- The API should provide clear and informative error messages.
- The end points should be intuitive and easy to use.

3.3.4 Reliability:

- The API should be available 99% of the time.
- The system should handle errors gracefully and provide appropriate responses.

3.4 Software Architecture:

3.4.1 Technology Stack:

- *Backend Framework*: Flask - for building the API.
- *Database*: SQLite - for storing user data.
- *Image Processing Library*: Pillow (PIL) - for performing image operations.
- *Request Handling: Flask*-RESTx - for managing API endpoints and documentation.

3.4.2 Components:

Authentication Module

- Handles user registration and API key generation.
- Validates API keys for each request.

Image Processing Module

- Contains functions for various image operations (crop, resize, rotate, filters).

Database Module

- Manages user data using SQL Alchemy ORM.

3.5 External Interfaces:

User Interface

- No direct user interface: users interact with the API through HTTP requests.

API Endpoints

- Detailed documentation of each endpoint, including required inputs and expected outputs, is provided using Flask-RESTx.

Database Interface

- SQLite database is used for storing user data, accessed through SQL Alchemy.

3.6 Constraints and Assumptions:

Constraints

- The system must run on a server with Python 3.x installed.
- The API should be compatible with common image formats (JPEG, PNG, GIF, BMP, TIFF).

Assumptions

- Users have valid API keys for accessing the endpoints.

- The server environment is secure and can handle the expected load.

3.7 Testing:

Unit Testing

- Each image processing function should have unit tests to ensure correct functionality.

Integration Testing

- Endpoints should be tested to verify that they work correctly with the authentication and image processing modules.

Performance Testing

- The API should be tested under load to ensure it can handle multiple requests efficiently.

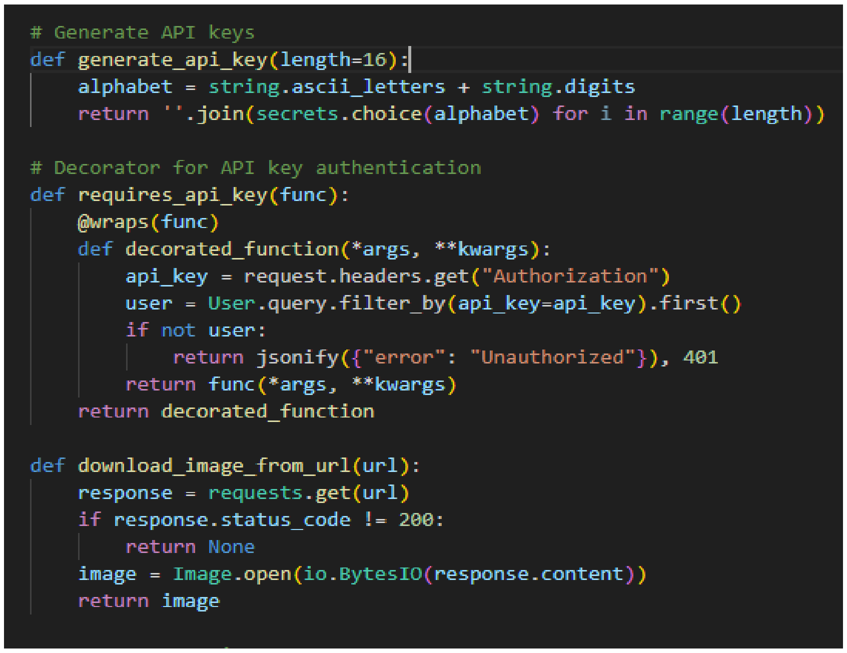
Chapter 4: Code Analysis and Evaluation

4.1 Structure and Organization:

The code follows a modular structure, separating concerns into different files and directories. It uses Flask-RESTx for API documentation and routing. The application is divided into two main namespaces: "auth" for user authentication operations and "image" for image processing operations.

4.1.1 User Authentication:

The code includes user authentication functionality, allowing users to register with a username and password. It generates a unique API key for each registered user, which is used for authentication in the image processing endpoints. The ‘*requires_api_key*’ decorator is used to protect the image processing endpoints, ensuring that only authenticated users can access them.



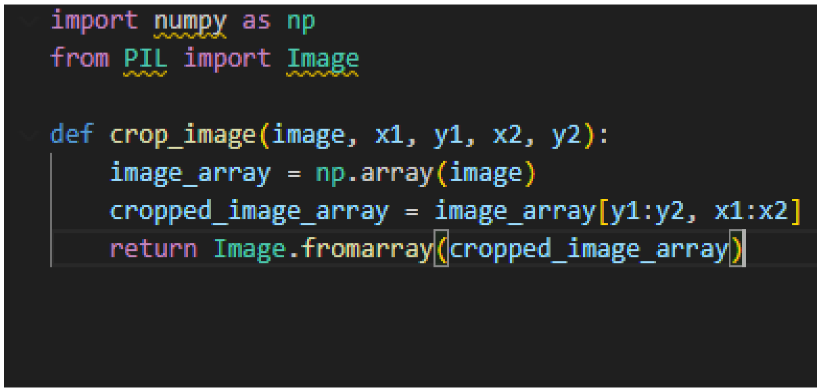
```python
# Generate API keys
def generate_api_key(length=16):
    alphabet = string.ascii_letters + string.digits
    return ''.join(secrets.choice(alphabet) for i in range(length))

# Decorator for API key authentication
def requires_api_key(func):
    @wraps(func)
    def decorated_function(*args, **kwargs):
        api_key = request.headers.get("Authorization")
        user = User.query.filter_by(api_key=api_key).first()
        if not user:
            return jsonify({"error": "Unauthorized"}), 401
        return func(*args, **kwargs)
    return decorated_function

def download_image_from_url(url):
    response = requests.get(url)
    if response.status_code != 200:
        return None
    image = Image.open(io.BytesIO(response.content))
    return image
```

Figure 12: Code for User Authentication

4.1.2 Image Processing Endpoints:

Each endpoint accepts either an image file or a URL as input and returns the processed image. The code uses the Pillow library for performing the actual image processing operations, details of the operation codes are as follows.

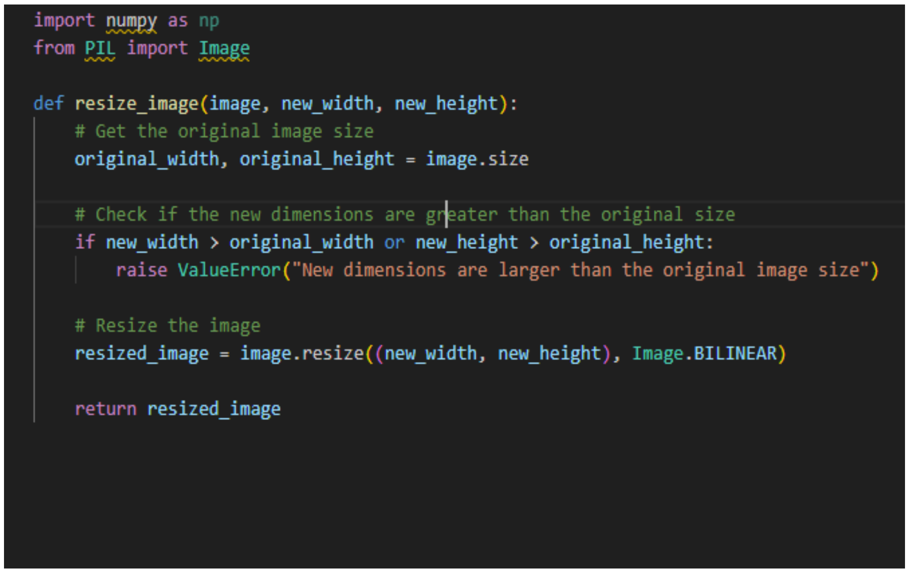
- **Cropping:**



```python
import numpy as np
from PIL import Image

def crop_image(image, x1, y1, x2, y2):
    image_array = np.array(image)
    cropped_image_array = image_array[y1:y2, x1:x2]
    return Image.fromarray(cropped_image_array)
```

Figure 13-a: Code for cropping

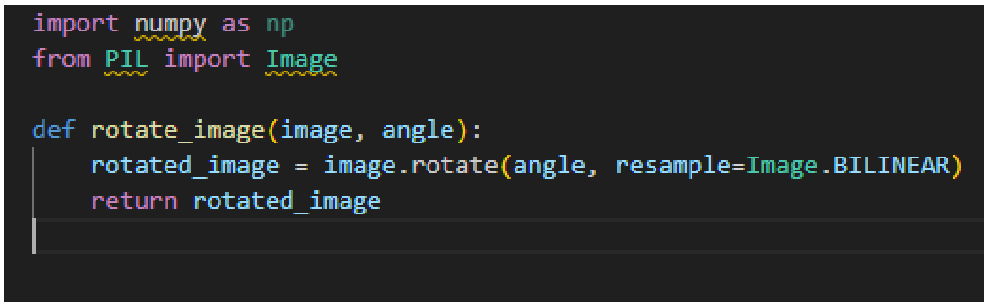
- **Resizing:**



```python
import numpy as np
from PIL import Image

def resize_image(image, new_width, new_height):
    # Get the original image size
    original_width, original_height = image.size

    # Check if the new dimensions are greater than the original size
    if new_width > original_width or new_height > original_height:
        raise ValueError("New dimensions are larger than the original image size")

    # Resize the image
    resized_image = image.resize((new_width, new_height), Image.BILINEAR)

    return resized_image
```

Figure 13-b: Code for resizing.

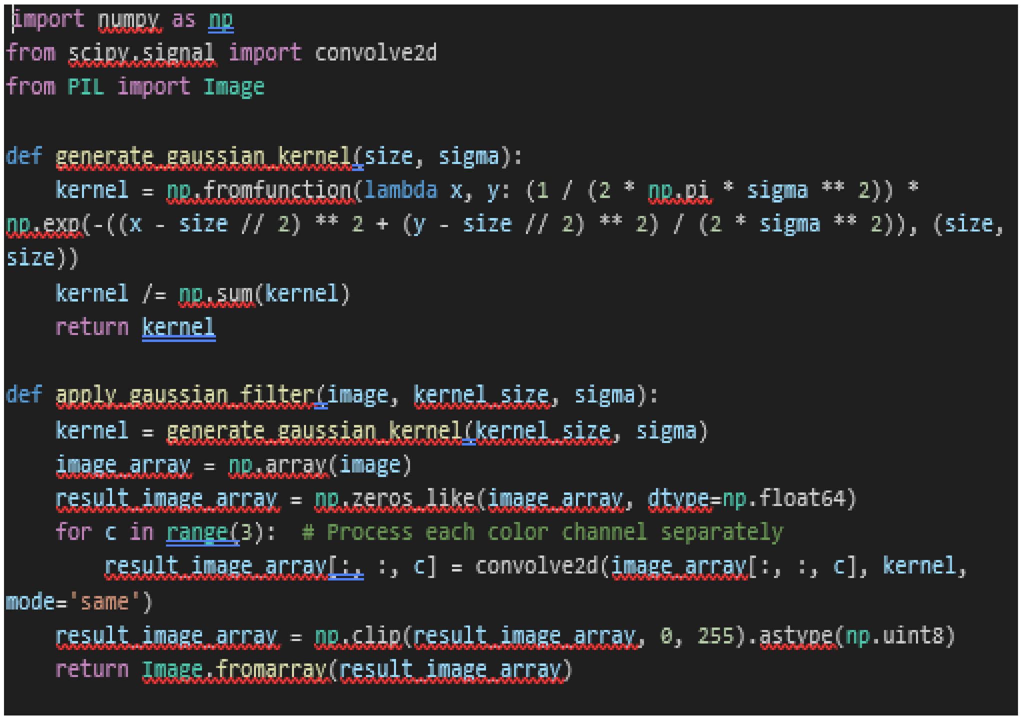
- **Rotating:**



```python
import numpy as np
from PIL import Image

def rotate_image(image, angle):
    rotated_image = image.rotate(angle, resample=Image.BILINEAR)
    return rotated_image
```

Figure 13-c: Code for rotating.

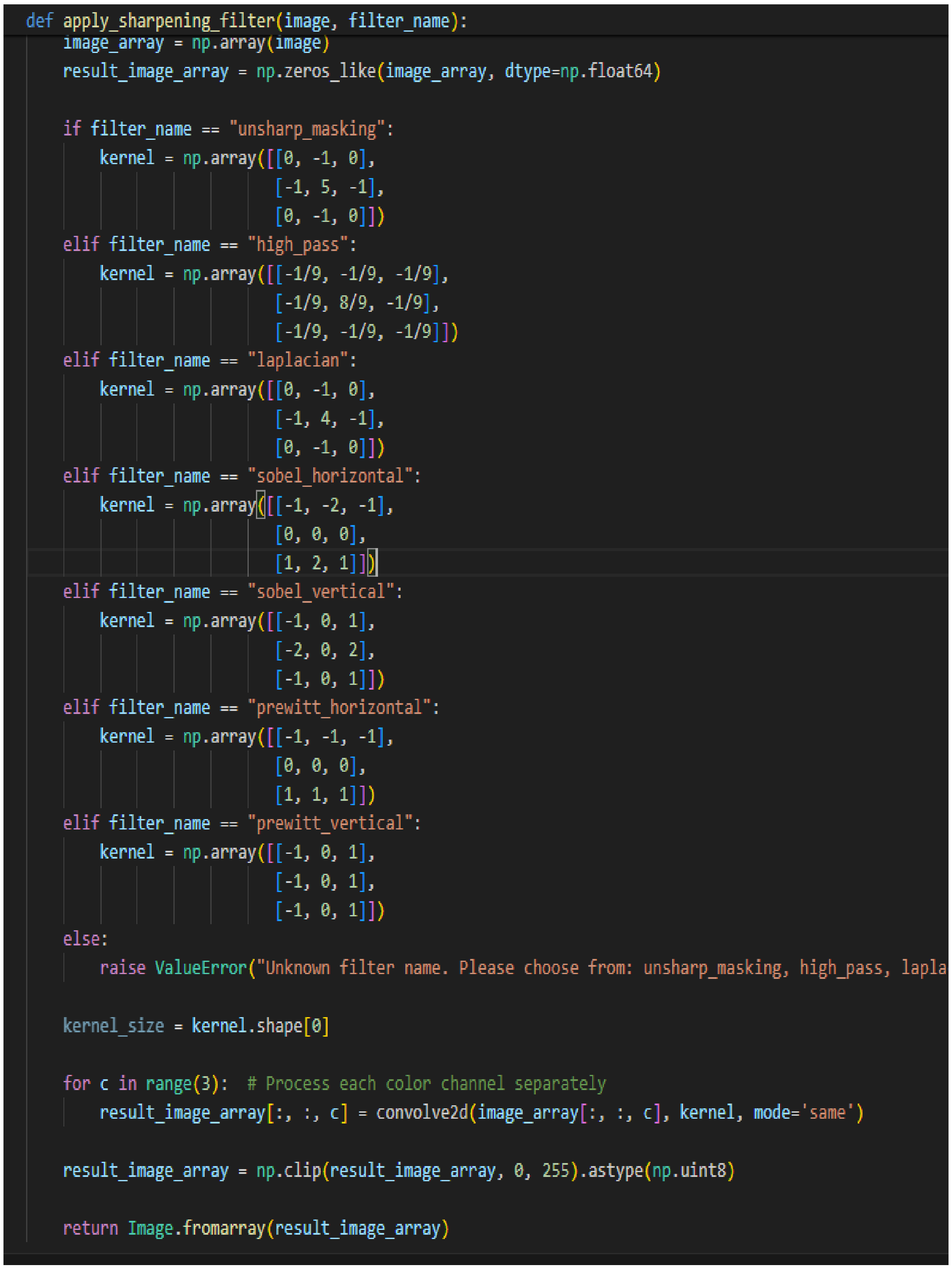
- **Gaussian:**



```python
import numpy as np
from scipy.signal import convolve2d
from PIL import Image

def generate_gaussian_kernel(size, sigma):
    kernel = np.fromfunction(lambda x, y: (1 / (2 * np.pi * sigma ** 2)) *
np.exp(-((x - size // 2) ** 2 + (y - size // 2) ** 2) / (2 * sigma ** 2)), (size,
size))
    kernel /= np.sum(kernel)
    return kernel

def apply_gaussian_filter(image, kernel_size, sigma):
    kernel = generate_gaussian_kernel(kernel_size, sigma)
    image_array = np.array(image)
    result_image_array = np.zeros_like(image_array, dtype=np.float64)
    for c in range(3):  # Process each color channel separately
        result_image_array[:, :, c] = convolve2d(image_array[:, :, c], kernel,
mode='same')
    result_image_array = np.clip(result_image_array, 0, 255).astype(np.uint8)
    return Image.fromarray(result_image_array)
```

Figure 13-d: Code for gaussian filter.

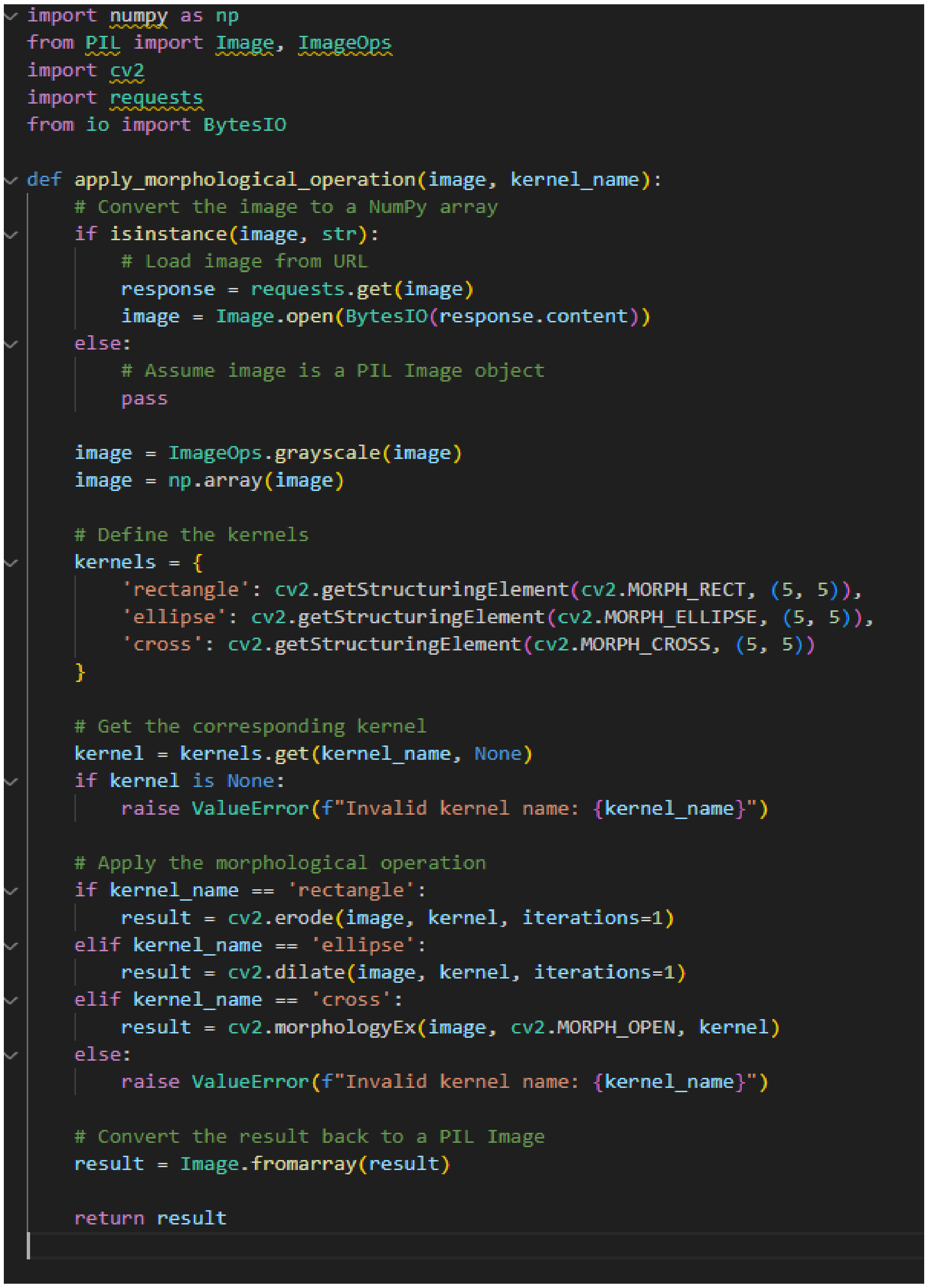
- **Sharpening:**



```python
def apply_sharpening_filter(image, filter_name):
    image_array = np.array(image)
    result_image_array = np.zeros_like(image_array, dtype=np.float64)

    if filter_name == "unsharp_masking":
        kernel = np.array([[0, -1, 0],
                           [-1, 5, -1],
                           [0, -1, 0]])
    elif filter_name == "high_pass":
        kernel = np.array([[-1/9, -1/9, -1/9],
                           [-1/9, 8/9, -1/9],
                           [-1/9, -1/9, -1/9]])
    elif filter_name == "laplacian":
        kernel = np.array([[0, -1, 0],
                           [-1, 4, -1],
                           [0, -1, 0]])
    elif filter_name == "sobel_horizontal":
        kernel = np.array([[-1, -2, -1],
                           [0, 0, 0],
                           [1, 2, 1]])
    elif filter_name == "sobel_vertical":
        kernel = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]])
    elif filter_name == "prewitt_horizontal":
        kernel = np.array([[-1, -1, -1],
                           [0, 0, 0],
                           [1, 1, 1]])
    elif filter_name == "prewitt_vertical":
        kernel = np.array([[-1, 0, 1],
                           [-1, 0, 1],
                           [-1, 0, 1]])
    else:
        raise ValueError("Unknown filter name. Please choose from: unsharp_masking, high_pass, lapla

    kernel_size = kernel.shape[0]

    for c in range(3):  # Process each color channel separately
        result_image_array[:, :, c] = convolve2d(image_array[:, :, c], kernel, mode='same')

    result_image_array = np.clip(result_image_array, 0, 255).astype(np.uint8)

    return Image.fromarray(result_image_array)
```

Figure 13-e: Code for sharpening filter.

- **Morphological:**



```python
import numpy as np
from PIL import Image, ImageOps
import cv2
import requests
from io import BytesIO

def apply_morphological_operation(image, kernel_name):
    # Convert the image to a NumPy array
    if isinstance(image, str):
        # Load image from URL
        response = requests.get(image)
        image = Image.open(BytesIO(response.content))
    else:
        # Assume image is a PIL Image object
        pass

    image = ImageOps.grayscale(image)
    image = np.array(image)

    # Define the kernels
    kernels = {
        'rectangle': cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)),
        'ellipse': cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5)),
        'cross': cv2.getStructuringElement(cv2.MORPH_CROSS, (5, 5))
    }

    # Get the corresponding kernel
    kernel = kernels.get(kernel_name, None)
    if kernel is None:
        raise ValueError(f"Invalid kernel name: {kernel_name}")

    # Apply the morphological operation
    if kernel_name == 'rectangle':
        result = cv2.erode(image, kernel, iterations=1)
    elif kernel_name == 'ellipse':
        result = cv2.dilate(image, kernel, iterations=1)
    elif kernel_name == 'cross':
        result = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
    else:
        raise ValueError(f"Invalid kernel name: {kernel_name}")

    # Convert the result back to a PIL Image
    result = Image.fromarray(result)

    return result
```

Figure 13-f: S Code for morphological operation.

4.2 Error Handling and Input Validation:

The code includes basic error handling and input validation, such as checking if a file or URL is provided and if the required parameters are present.

If an error occurs, the application returns a JSON response with an error message and an appropriate HTTP status code.

4.3 Database Integration:

The code uses Flask-SQL Alchemy to integrate a SQLite database for storing user information. The User model is defined to store the username, password, and API key for each registered user.

4.4 Documentation of APIs:

Here's the API documentation for the Flask application:

- **Image Processing API:**

  This API allows users to perform various image processing operations.

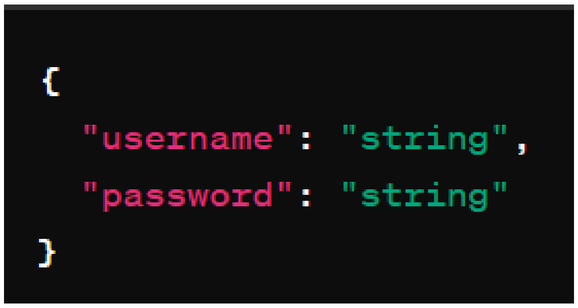
  Authentication:

  Endpoint: "/auth/register"

  Method: POST

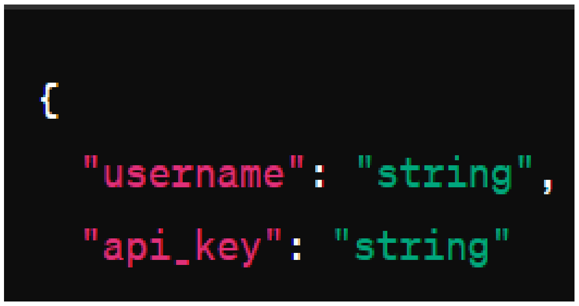
  Description: Register a new user to obtain an API key.

  Request Body



```
{
  "username": "string",
  "password": "string"
}
```

Response Body



```
{
  "username": "string",
  "api_key": "string"
```

- **Image Processing Operations:**

Note: All endpoints require API key authentication using the 'Authorization' header.

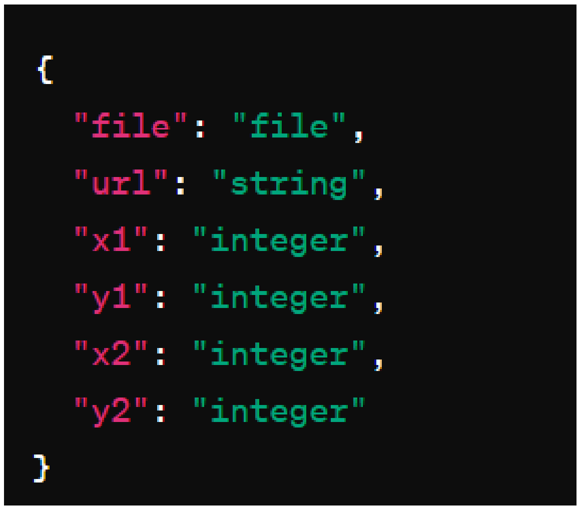
<u>*Crop Image:*</u>

Endpoint: "/crop"

Method: POST

Description: Crop an image to the specified dimensions.

Request Body



```
{
  "file": "file",
  "url": "string",
  "x1": "integer",
  "y1": "integer",
  "x2": "integer",
  "y2": "integer"
}
```

Response

Returns the cropped image in the original format.

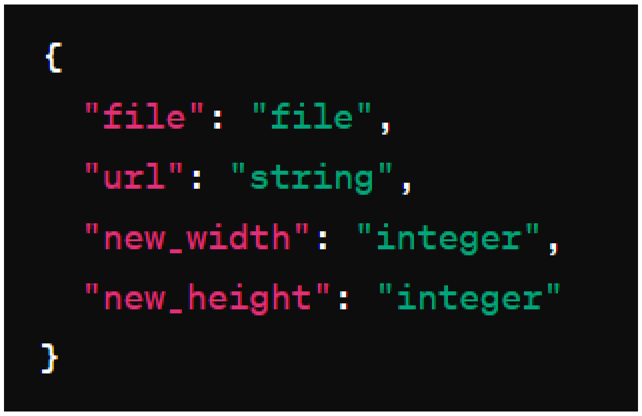
Resize Image:

Endpoint: “/resize”

Method: POST

Description: Resize an image to the specified dimensions.

Request Body



```
{
  "file": "file",
  "url": "string",
  "new_width": "integer",
  "new_height": "integer"
}
```

Response

Returns the resized image in the original format.

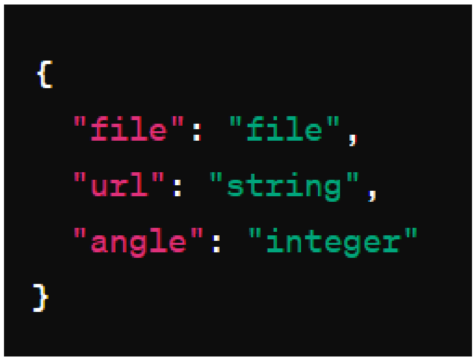
Rotate Image:

Endpoint: /rotate

Method: POST

Description: Rotate an image on the specified angle.

Request Body



```
{
  "file": "file",
  "url": "string",
  "angle": "integer"
}
```

Response

Returns the rotated image in the original format.

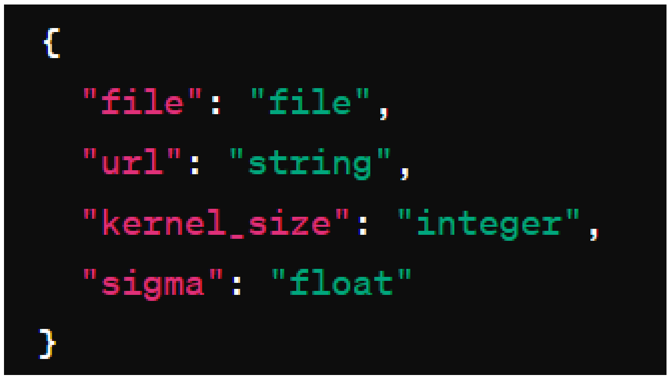
Apply Gaussian Filter:

Endpoint: “/gaussian_filter”

Method: POST

Description: Apply a Gaussian filter to the image.

Request Body



```
{
  "file": "file",
  "url": "string",
  "kernel_size": "integer",
  "sigma": "float"
}
```

Response

Returns the filtered image in the original format.

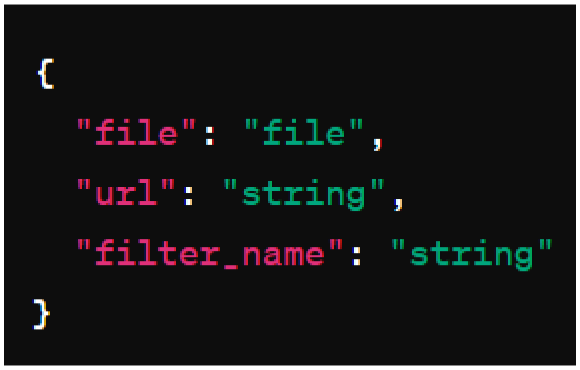
Apply Sharpening Filter:

Endpoint: “/sharpening_filter”

Method: POST

Description: Apply a sharpening filter to the image.

Request Body



```
{
  "file": "file",
  "url": "string",
  "filter_name": "string"
}
```

Response

Returns the filtered image in the original format.

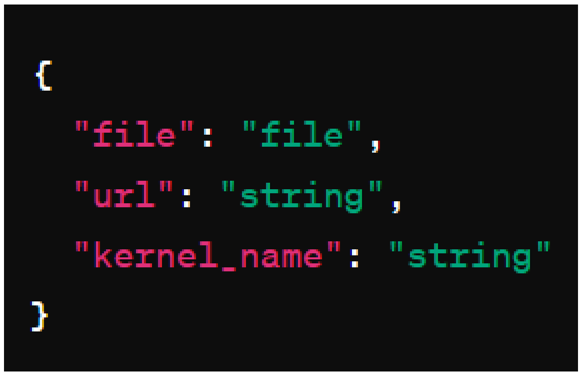
<u>Apply Morphological Operation:</u>

Endpoint: “/morphological_operation”

Method: POST

Description: Apply a morphological operation to the image.

Request Body



```
{
  "file": "file",
  "url": "string",
  "kernel_name": "string"
}
```

Response

Returns the processed image in the original format.

4.5 Testing:

Testing is done on two types,

- Unit Testing: Variables and Functions are tested through python library.
- API Testing done through Postman: The API endpoints are verified either they are working properly or not.

4.5.1 Unit Testing:

An example code for unit test of cropping function is provided below.

```
{
import unittest
from PIL import Image
from crop_operations import crop_image


class TestCropImage(unittest.TestCase):
def setUp(self):
# Create a sample image for testing
self.image = Image.new('RGB', (100, 100), 'white')


def test_crop_image(self):
cropped_image = crop_image(self.image, 10, 10, 50, 50)
self.assertEqual(cropped_image.size, (40, 40))


def test_invalid_coordinates(self):
with self.assertRaises(ValueError):
```

crop_image(self.image, -10, -10, 110, 110)

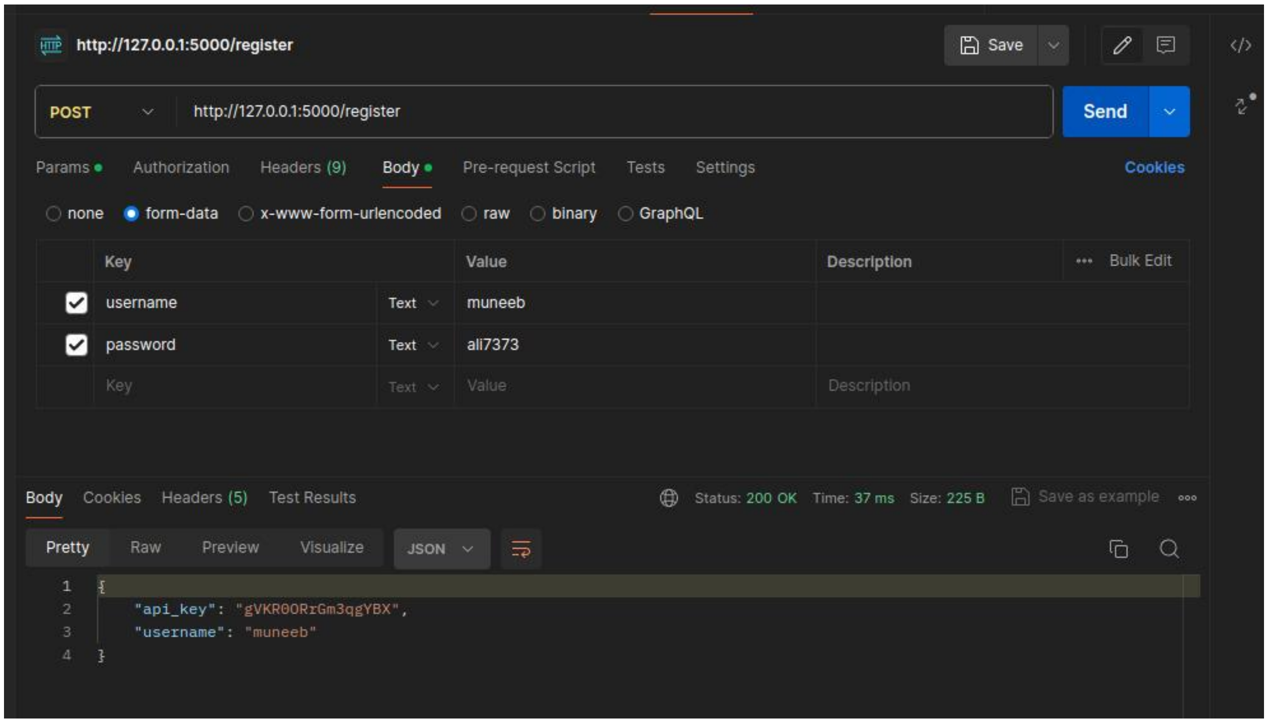
if __name__ == '__main__':

unittest.main()

}

4.5.1 API Testing:

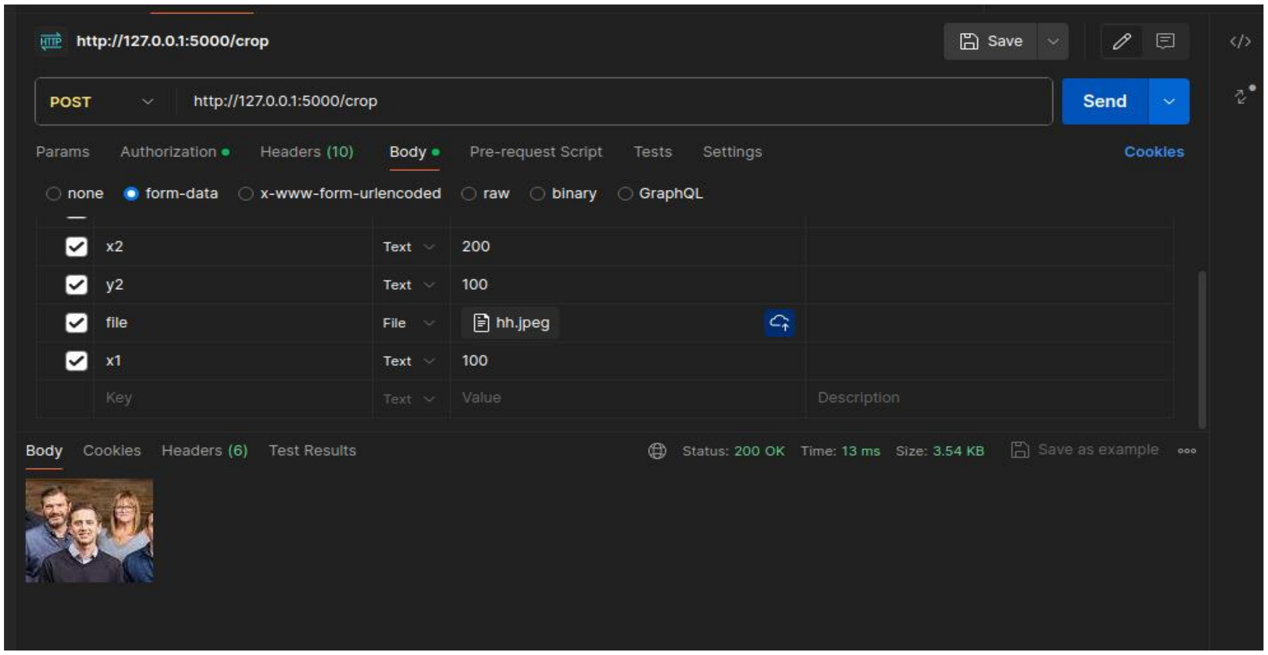
Below is shown the snip-shot of various API endpoints like registration, cropping, rotating.



Key		Value	Description
username	Text	muneeb	
password	Text	ali7373	

```
{
    "api_key": "gVKR0ORrGm3qgYBX",
    "username": "muneeb"
}
```

Figure 14-a: API Testing for Registration.



http://127.0.0.1:5000/crop

POST http://127.0.0.1:5000/crop

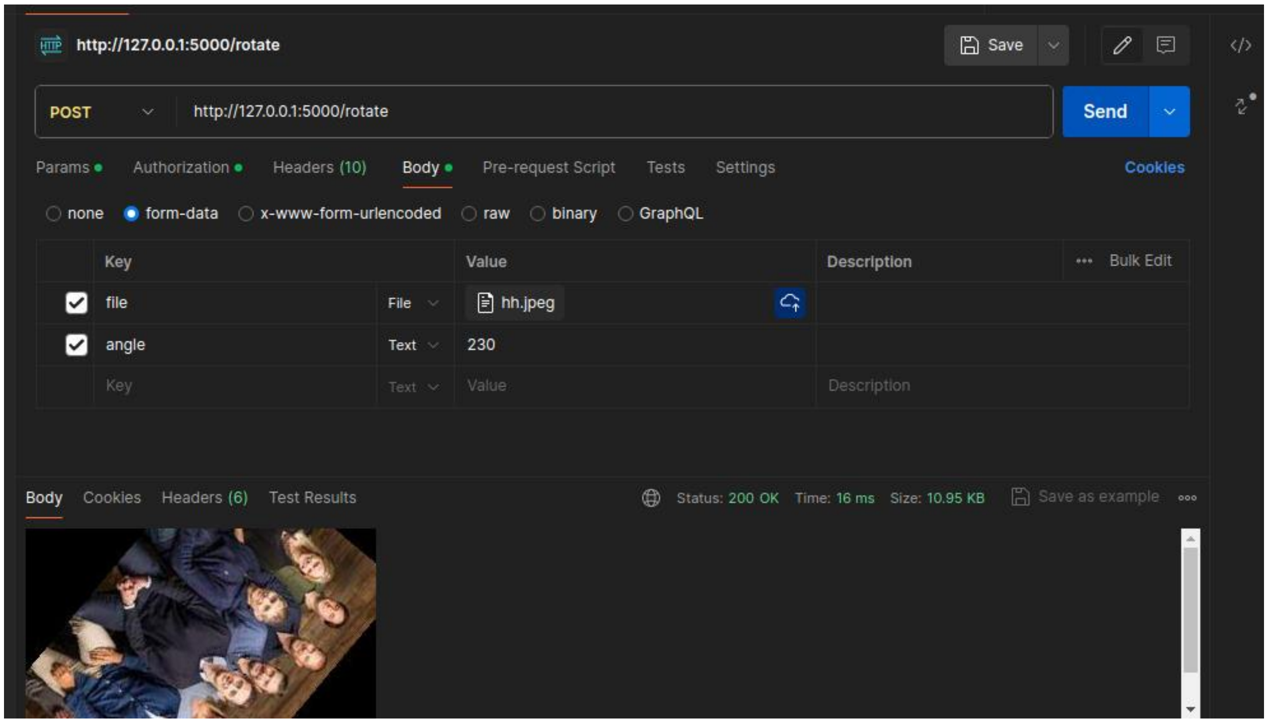
Params | Authorization | Headers (10) | Body | Pre-request Script | Tests | Settings | Cookies

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	Key	Type	Value	Description
✓	x2	Text	200	
✓	y2	Text	100	
✓	file	File	hh.jpeg	
✓	x1	Text	100	
	Key	Text	Value	Description

Body | Cookies | Headers (6) | Test Results — Status: 200 OK Time: 13 ms Size: 3.54 KB Save as example

Figure 14-b: API Testing for Cropping.



http://127.0.0.1:5000/rotate

POST http://127.0.0.1:5000/rotate

Params | Authorization | Headers (10) | Body | Pre-request Script | Tests | Settings | Cookies

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	Key	Type	Value	Description
✓	file	File	hh.jpeg	
✓	angle	Text	230	
	Key	Text	Value	Description

Body | Cookies | Headers (6) | Test Results — Status: 200 OK Time: 16 ms Size: 10.95 KB Save as example

Figure 14-c: API Testing for Rotating.

Chapter 5: Conclusion

In this thesis, we discussed a on demand image rescaling system that can provide various image manipulation techniques on user requests smartly and more efficiently than the typically used software. Our proposed system’s advantage over other systems lies in its versatility, security features, and user-friendly design. The techniques used in our proposed system; Image Processing techniques to manipulate images; are also briefly explained, including their working and importance. The increasing use of digital platforms have fast, visual rich and appealing images that need to be either manipulated or optimized based on user requirement. For the specific purpose our system has provided a robust design. Additionally, the objectives; API design and image operations, are attained. Hence provides quality visuals to the users.

Documentation and Testing of the APIs (through Postman) is shown that would be beneficial for the developers. The same APIs can be used to serve marketing business purposes. Only the integration of a specific front end would be required.

Our proposed system, ODIR, is cost-effective as it is purely made for the core purpose of serving Pakistan, any system that could be beneficial to its citizens. Else, similar solutions provided by other countries (cloudinary) are very costly. Moreover, ODIR provides an ease to adoption which can be adopted by beneficiaries, mass deployment can also be done and most importantly no product training is required.

Chapter 6: Future Work

Future milestones that need to be achieved to commercialize this project are the following.

6.1 Intelligent Cropping:

Our next main objective is to utilize advanced machine learning algorithms to intelligently identify, and crop images based on content. It will prevent unintentional misinformation by ensuring accurate representation.

Recognizing specific objects in images will enable us automated cropping, adjustment based on subjects.

6.2 User Interface:

A prototype interface has been developed. Integration of the developed APIs and more to come will be done in the next steps. Improvement in performance, aesthetics and user engaging front end would be the next priority.

6.3 Bulk Image Resizing:

Bulk image-resizing techniques and tools enable you to resize images simultaneously, significantly reducing the time spent manually resizing one image at a time. This will also be one of the main steps taken to improve the existing demo.

References and Work Cited

1. *Thumbor*, an open-source smart imaging service, offers valuable capabilities for resizing, cropping, and optimizing images dynamically. [online] Available at <https://www.thumbor.org>

2. The article "*Improving Performance & Perception: On-Demand Image Resizing*" from Site Point. [online] Available at: <https://www.sitepoint.com/improving-performance-perception-on-demand-image-resizing/>

3. Entitled Digital Image Processing, Fourth Edition, ISBN 978-0-13-335672-4, by Rafael C. Gonzalez and Richard E. Woods, published by Pearson Education © 2018, p.165-181.

4. Adobe photoshop, a digital image editing that runs on monthly subscription model. [online] Available at <https://www.adobe.com/products/photoshop.html>

5. A collection of command-line tools that can be used to modify and manipulate images. [online] Available at <https://imagemagick.org/script/command-line-tools.php>

6. A study between OpenCV and MATLAB's different image processing built in libraries Section 5. [online] Available at <https://www.researchgate.net/publication/332873639_Matlab_vs_OpenCV_A_Comparative_Study_of_Different_Machine_Learning_Algorithms>

7. https://www.mathworks.com/help/images/

8. "Research on Digital Image Processing Technology and Its Application" by Zhou et al. (2018). *Advances in Intelligent Systems Research,* , 8th International Conference on

Management, Education, and Information (MEICI 2018) vol 163 no.5, p.587-591. Available at: <https://www.atlantis-press.com/article/55910087.pdf>

9. Neeraj Kumar Pandey; Manoj Diwakar, (2020). A Review of Cloud based Image Processing Services. 2020 7th International Conference on Computing for Sustainable Global Development (INDIACom) p 5005-5010. Available at: <https://ieeexplore.ieee.org/document/7354081>
10. "Comparative Analysis of Image Processing Techniques Using Cloud Computing Paradigm". *Published in* 2017 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT) p525-530. Available at: <https://ieeexplore.ieee.org/document/8342970>