Flask Image Processing Application

Overview

This Flask application allows users to upload images, generate color histograms, create segmentation masks, and perform image manipulation tasks such as resizing, cropping, and format conversion.

Prerequisites

Python 3.x
Flask
Pillow
OpenCV
Matplotlib

Configuration

Ensure the following folders are present in your project directory:
- `uploads`: To store uploaded images.
- `results`: To store processed images.

Application Structure

app.py

This is the main application file that contains all the routes and image processing logic.

Routes

GET /: Renders the main HTML page.
POST /upload: Handles image upload and saves files to the upload folder.
GET /uploads/<filename>: Returns the uploaded file.
GET /histogram/<filename>: Generates and returns a color histogram for the given image.
GET /segmentation/<filename>: Generates and returns a segmentation mask for the given image.
POST /resize/<filename>: Resizes the image to specified dimensions and returns the result.
POST /crop/<filename>: Crops the image to specified coordinates and returns the result.
POST /convert/<filename>: Converts the image to the specified format and returns the result.

Image Upload and Storage with Batch Processing

Code Snippet: Image Upload and Storage

```
@app.route('/upload', methods=['POST'])
def upload_image():
    if 'images' not in request.files:
        return jsonify({"error": "No images provided"}), 400
```

```
    files = request.files.getlist('images')
    file_info = []

    for file in files:
        if file.filename == '':
            return jsonify({"error": "No selected file"}), 400
        if file:
            filename = secure_filename(file.filename)
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(file_path)
            with Image.open(file_path) as img:
                width, height = img.size
            file_info.append({
                "filename": filename,
                "width": width,
                "height": height,
                "path": url_for('uploaded_file', filename=filename),
                "mode": img.mode,
                "format": img.format
            })

    return jsonify({"message": "Images successfully uploaded", "files": file_info}), 200
```

Explanation: This function handles the upload of images. It checks if images are provided, iterates through the list of uploaded images, saves them to the uploads directory, and returns their details such as filename, width, height, mode, and format.

Generating Color Histograms and Segmentation Masks

Code Snippet: Generating Color Histogram

```
def generate_histogram(image_path):
    image = cv2.imread(image_path)
    color = ('b', 'g', 'r')
    figure = Figure()
    axis = figure.add_subplot(1, 1, 1)

    for i, col in enumerate(color):
        hist = cv2.calcHist([image], [i], None, [256], [0, 256])
        axis.plot(hist, color=col)
```

```
    output_path = os.path.join(app.config['RESULT_FOLDER'], os.path.basename(image_path) +
'_histogram.png')
    figure.savefig(output_path)

    return output_path

@app.route('/histogram/<filename>', methods=['GET'])
def get_histogram(filename):
    image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    histogram_path = generate_histogram(image_path)
    return send_file(histogram_path, mimetype='image/png')
```

Explanation: The generate_histogram function reads the image, calculates the histogram for each color channel (blue, green, and red), plots these histograms, and saves the plot as an image. The get_histogram route uses this function to generate the histogram for a given image filename and returns the histogram image.

Code Snippet: Generating Segmentation Mask

```
def generate_segmentation_mask(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, mask = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    output_path = os.path.join(app.config['RESULT_FOLDER'], os.path.basename(image_path) +
'_segmentation.png')
    cv2.imwrite(output_path, mask)

    return output_path

@app.route('/segmentation/<filename>', methods=['GET'])
def get_segmentation_mask(filename):
    image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    mask_path = generate_segmentation_mask(image_path)
    return send_file(mask_path, mimetype='image/png')
```

Explanation: The generate_segmentation_mask function converts the image to grayscale and then applies a binary threshold to create a segmentation mask. The get_segmentation_mask route generates and returns this mask for a given image filename.

Image Manipulation Tasks: Resizing, Cropping, and Format Conversion

Code Snippet: Resizing Image

```
@app.route('/resize/<filename>', methods=['POST'])
def resize_image(filename):
    image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    image = Image.open(image_path)
    width = request.json.get('width')
    height = request.json.get('height')
    resized_image = image.resize((width, height))

    output = io.BytesIO()
    resized_image.save(output, format='PNG')
    output.seek(0)

    return send_file(output, mimetype='image/png')
```

Explanation: The resize_image function resizes the image to the specified width and height. The resized image is then returned as a PNG file. The route reads the width and height from the POST request's JSON data.

Code Snippet: Cropping Image

```
@app.route('/crop/<filename>', methods=['POST'])
def crop_image(filename):
    image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    image = Image.open(image_path)
    left = request.json.get('left')
    top = request.json.get('top')
    right = request.json.get('right')
    bottom = request.json.get('bottom')
```

```
    if left < 0 or top < 0 or right > image.width or bottom > image.height or left >= right or top >=
bottom:
        return jsonify({"error": "Invalid crop coordinates"}), 400

    cropped_image = image.crop((left, top, right, bottom))

    output = io.BytesIO()
    cropped_image.save(output, format='PNG')
    output.seek(0)

    return send_file(output, mimetype='image/png')
```

Explanation: The crop_image function crops the image based on the specified coordinates (left, top, right, bottom). It checks if the coordinates are valid and returns the cropped image as a PNG file. The route reads the coordinates from the POST request's JSON data.

Code Snippet: Converting Image Format

```
@app.route('/convert/<filename>', methods=['POST'])
def convert_image(filename):
    image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    image = Image.open(image_path)
    format = request.json.get('format').lower()

    valid_formats = ["jpeg", "png", "bmp", "gif", "tiff"]
    if format not in valid_formats:
        return jsonify({"error": f"Unsupported format: {format}"}), 400

    output_path = os.path.join(app.config['RESULT_FOLDER'],
os.path.splitext(os.path.basename(image_path))[0] + f'.{format}')
    image.save(output_path, format=format.upper())
    return send_file(output_path, mimetype=f'image/{format}')
```

Explanation: The convert_image function converts the image to the specified format. It checks if the format is valid, converts the image, saves it, and returns the converted image. The route reads the format from the POST request's JSON data.

---

templates/index.html

The main HTML file for the front-end interface, providing upload functionality and buttons to trigger various image processing tasks.

static/styles.css

Contains the CSS for styling the application.

static/scripts.js

Contains JavaScript functions to handle form submissions and fetch responses from the server.

Running the Application

1. Activate the virtual environment:
```
source venv/bin/activate   # On Windows: venv\Scripts\activate
```

2. Run the Flask application:
```
python app.py
```

3. Open your web browser and navigate to `http://127.0.0.1:5000/`.

Usage

Uploading Images

1. Overview: This function allows users to upload multiple images at once.
2. Steps:
    - Click on the "Choose files" button and select one or more images to upload.
    - Click on the "Upload" button to upload the selected images.
3. Algorithm Explanation:
    - The images are saved to the server.
    - The details of each image, such as filename, width, height, mode, and format, are returned.

Generating Color Histograms

1. Overview: This function generates a color histogram for the uploaded image.
2. Steps:
    - Enter the filename of the uploaded image in the input box.
    - Click on the "Get Histogram" button to generate and display the histogram.
3. Algorithm Explanation:
    - The image is read, and histograms for the blue, green, and red channels are calculated.
    - These histograms are plotted and saved as an image.

Generating Segmentation Masks

1. Overview: This function generates a segmentation mask for the uploaded image.
2. Steps:

- Enter the filename of the uploaded image in the input box.
- Click on the "Get Segmentation Mask" button to generate and display the segmentation mask.

3. Algorithm Explanation:
    - The image is converted to grayscale.
    - A binary threshold is applied to create a segmentation mask.
    - The mask is saved as an image.

Resizing Images

1. Overview: This function resizes the uploaded image to specified dimensions.
2. Steps:
    - Enter the filename of the uploaded image in the input box.
    - Enter the desired width and height in the respective input boxes.
    - Click on the "Resize" button to resize and display the image.
3. Algorithm Explanation:
    - The image is resized based on the specified width and height.
    - The resized image is returned as a PNG file.

Cropping Images

1. Overview: This function crops the uploaded image to specified coordinates.
2. Steps:
    - Enter the filename of the uploaded image in the input box.
    - Enter the desired crop coordinates (left, top, right, bottom) in the respective input boxes.
    - Click on the "Crop" button to crop and display the image.
3. Algorithm Explanation:

- The image is cropped based on the specified coordinates.
- The cropped image is returned as a PNG file.

Converting Image Formats

1. Overview: This function converts the uploaded image to a specified format.
2. Steps:
    - Enter the filename of the uploaded image in the input box.
    - Select the desired format from the dropdown menu.
    - Click on the "Convert" button to convert and display the image.
3. Algorithm Explanation:
    - The image is converted to the specified format.
    - The converted image is saved and returned.