

## Assignment

**Problem Statement: Social media filters by applying Gaussian Blur, Edge Detection, Color Correction and Enhancement using OpenCV.**

**Required Libraries:** Flask, opencv-python, numpy

**Front-end/User-Interface Code:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Image Filter WebApp</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 40px;
      background-color: #f5f5f5;
      color: #333;
    }
    h2, h3 {
      text-align: center;
    }
    form {
      background: white;
      padding: 20px;
      max-width: 500px;
      margin: 20px auto;
      border-radius: 10px;
      box-shadow: 0px 0px 10px rgba(0,0,0,0.1);
    }
    input[type="file"], select, button {
      width: 100%;
      padding: 10px;
      margin-top: 10px;
      margin-bottom: 20px;
      border: 1px solid #ccc;
      border-radius: 5px;
    }
```

```
img {
  display: block;
  margin: 20px auto;
  max-width: 300px;
  border-radius: 10px;
  box-shadow: 0px 0px 5px rgba(0,0,0,0.2);
}
.download-btn {
  display: block;
  width: 200px;
  margin: 20px auto;
  text-align: center;
  background-color: green;
  color: white;
  padding: 10px;
  text-decoration: none;
  border-radius: 5px;
}
.download-btn:hover {
  background-color: darkgreen;
}
</style>
</head>

<body>

<h2>🎨 Upload an Image and Apply Filters</h2>

<form action="/" method="POST" enctype="multipart/form-data">
  <label>Upload Image:</label>
  <input type="file" name="file" required>

  {% if uploaded_image %}
  <h3>Uploaded Image:</h3>
  
  {% endif %}

  <label>Select a Filter:</label>
  <select name="filter" required>
    <optgroup label="Blurring Filters">
```

```
<option value="gaussian_blur">Gaussian Blur</option>
<option value="average_blur">Average Blur</option>
<option value="median_blur">Median Blur</option>
<option value="bilateral_filter">Bilateral Filter</option>
</optgroup>

<optgroup label="Edge Detection">
  <option value="canny_edge">Canny Edge Detection</option>
  <option value="sobel_edge">Sobel Edge Detection</option>
  <option value="laplacian_edge">Laplacian Edge Detection</option>
</optgroup>

<optgroup label="Thresholding">
  <option value="otsu_threshold">Otsu Thresholding</option>
</optgroup>

<optgroup label="Morphological Operations">
  <option value="dilation">Dilation</option>
  <option value="erosion">Erosion</option>
</optgroup>

<optgroup label="Color Processing">
  <option value="convert_ycbcr">Convert to YCbCr</option>
  <option value="convert_hsv">Convert to HSV</option>
  <option value="pseudo_color">Pseudo Color Mapping</option>
</optgroup>

<optgroup label="Stylization">
  <option value="stylization">Stylization</option>
  <option value="pencil_sketch">Pencil Sketch</option>
  <option value="cartoon">Cartoon Effect</option>
</optgroup>

<optgroup label="Feature Extraction">
  <option value="contours">Contour Detection</option>
</optgroup>
</select>

<button type="submit">Apply Selected Filter</button>
</form>
```

```
{% if processed_image %}
<h3>🖼️ Filtered Image:</h3>

<a href="{{ processed_image }}" download class="download-btn">Download Filtered
Image</a>
{% endif %}

</body>
</html>
```

**Front-end/User-Interface File Link: - [index.html](#)**

**Back-end/Python Code:**

```
from flask import Flask, render_template, request
import cv2
import numpy as np
import os

app = Flask(__name__)
UPLOAD_FOLDER = "static/uploads"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route("/", methods=["GET", "POST"])
def index():
    uploaded = False
    current_image_path = os.path.join(UPLOAD_FOLDER, "current_upload.jpg")

    if request.method == "POST":
        if "file" in request.files:
            file = request.files["file"]
            if file.filename != "":
                file.save(current_image_path)
                uploaded = True

    filter_type = request.form.get("filter")

    if os.path.exists(current_image_path) and filter_type:
        image = cv2.imread(current_image_path)
```

```

if filter_type == "gaussian_blur":
    processed_image = cv2.GaussianBlur(image, (15, 15), 0)
elif filter_type == "average_blur":
    processed_image = cv2.blur(image, (15, 15))
elif filter_type == "median_blur":
    processed_image = cv2.medianBlur(image, 15)
elif filter_type == "bilateral_filter":
    processed_image = cv2.bilateralFilter(image, 15, 75, 75)
elif filter_type == "canny_edge":
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    processed_image = cv2.Canny(gray, 100, 200)
elif filter_type == "sobel_edge":
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    processed_image = cv2.Sobel(gray, cv2.CV_64F, 1, 1, ksize=5)
    processed_image = cv2.convertScaleAbs(processed_image)
elif filter_type == "laplacian_edge":
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    processed_image = cv2.Laplacian(gray, cv2.CV_64F)
    processed_image = cv2.convertScaleAbs(processed_image)
elif filter_type == "sharpen":
    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    processed_image = cv2.filter2D(image, -1, kernel)
elif filter_type == "otsu_threshold":
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, processed_image = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
elif filter_type == "dilation":
    kernel = np.ones((5,5), np.uint8)
    processed_image = cv2.dilate(image, kernel, iterations=1)
elif filter_type == "erosion":
    kernel = np.ones((5,5), np.uint8)
    processed_image = cv2.erode(image, kernel, iterations=1)
elif filter_type == "convert_ycbcr":
    processed_image = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
elif filter_type == "convert_hsv":
    processed_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
elif filter_type == "pseudo_color":
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    processed_image = cv2.applyColorMap(gray, cv2.COLORMAP_JET)
elif filter_type == "stylization":

```

```
    processed_image = cv2.stylization(image, sigma_s=60, sigma_r=0.6)
elif filter_type == "pencil_sketch":
    gray, _ = cv2.pencilSketch(image, sigma_s=60, sigma_r=0.07, shade_factor=0.05)
    processed_image = gray
elif filter_type == "cartoon":
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.medianBlur(gray, 5)
    edges = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                  cv2.THRESH_BINARY, 9, 9)
    color = cv2.bilateralFilter(image, 9, 300, 300)
    processed_image = cv2.bitwise_and(color, color, mask=edges)
elif filter_type == "contours":
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 127, 255, 0)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    processed_image = cv2.drawContours(image.copy(), contours, -1, (0,255,0), 2)
else:
    processed_image = image

    processed_img_path = os.path.join(UPLOAD_FOLDER, "processed.jpg")
    cv2.imwrite(processed_img_path, processed_image)

    return render_template("index.html",
uploaded_image="static/uploads/current_upload.jpg",
processed_image="static/uploads/processed.jpg")

    return render_template("index.html", uploaded_image=None, processed_image=None)

if __name__ == "__main__":
    app.run(debug=True)
```

**Back-end/Python File: - [app.py](#)**

**Command for Running Python Flask Application: - python app.py**

**Folder Setup: -**

**image\_filter\_webapp(main folder)**

|→**templates(sub-folder)** → index.html(*Front-end/User-Interface File*)

|→app.py(*Back-end/Python file*)


**Image Filter Webapp Output Photos -  
Input Image:**



**Output: Canny Edge Detection**



### Before Applying Filter:

 Upload an Image and Apply Filters


Upload Image:

Watch.jpg

Select a Filter:

Canny Edge Detection


### Output file: Canny Edge Detection

 Upload an Image and Apply Filters

Upload Image:


No file chosen


Uploaded Image:



Select a Filter:

Gaussian Blur

 Filtered Image:






**Input Image:**



**Output: Pencil Sketch**



**Before Applying Filter:**


 **Upload an Image and Apply Filters**

Upload Image:

Watch.jpg

Select a Filter:


**Output file: Pencil Sketch**

 **Upload an Image and Apply Filters**


Upload Image:

No file chosen

**Uploaded Image:**



Select a Filter:

 **Filtered Image:**

