Assignment

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

Required Libraries: Flask, opency-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>
  <img src="{{ uploaded image }}" alt="Uploaded Image">
  {% endif %}
  <label>Select a Filter:</label>
  <select name="filter" required>
   <optgroup label="Blurring Filters">
```

```
<option value="gaussian blur">Gaussian Blur
   <option value="average blur">Average Blur</option>
   <option value="median blur">Median Blur
   <option value="bilateral filter">Bilateral Filter
  </optgroup>
  <optgroup label="Edge Detection">
   <option value="canny edge">Canny Edge Detection</option>
   <option value="sobel edge">Sobel Edge Detection
   <option value="laplacian edge">Laplacian Edge Detection/option>
  </optgroup>
  <optgroup label="Thresholding">
   <option value="otsu threshold">Otsu Thresholding
  </optgroup>
  <optgroup label="Morphological Operations">
   <option value="dilation">Dilation
   <option value="erosion">Erosion
  </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
  </optgroup>
  <optgroup label="Stylization">
   <option value="stylization">Stylization
   <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
</form>
```

```
{% if processed image %}
  <h3> Filtered Image:</h3>
  <img src="{{ processed image }}" alt="Processed Image">
  <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)
   \rightarrowapp.py(Back-end/Python file)
```

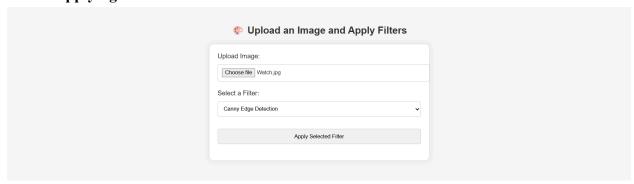
Image Filter Webapp Output Photos - Input Image:



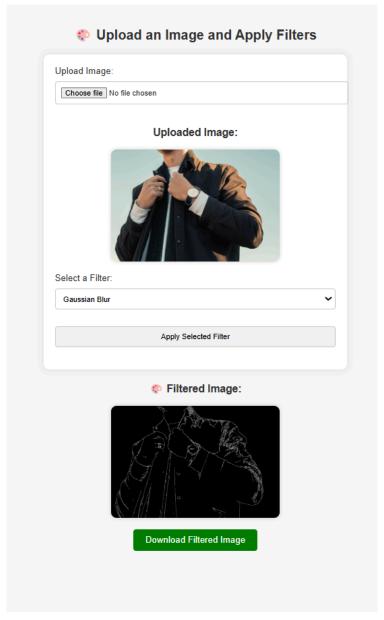
Output: Canny Edge Detection



Before Applying Filter:



Output file: Canny Edge Detection



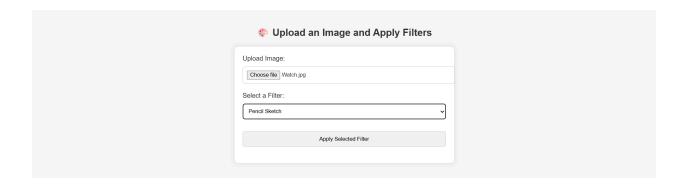
Input Image:



Output: Pencil Sketch



Before Applying Filter:



Output file: Pencil Sketch

