



Published in Towards Data Science



Hemanth Nag

Follow



May 29, 2021 · 5 min read · Listen



Save



Camera App with Flask and OpenCV

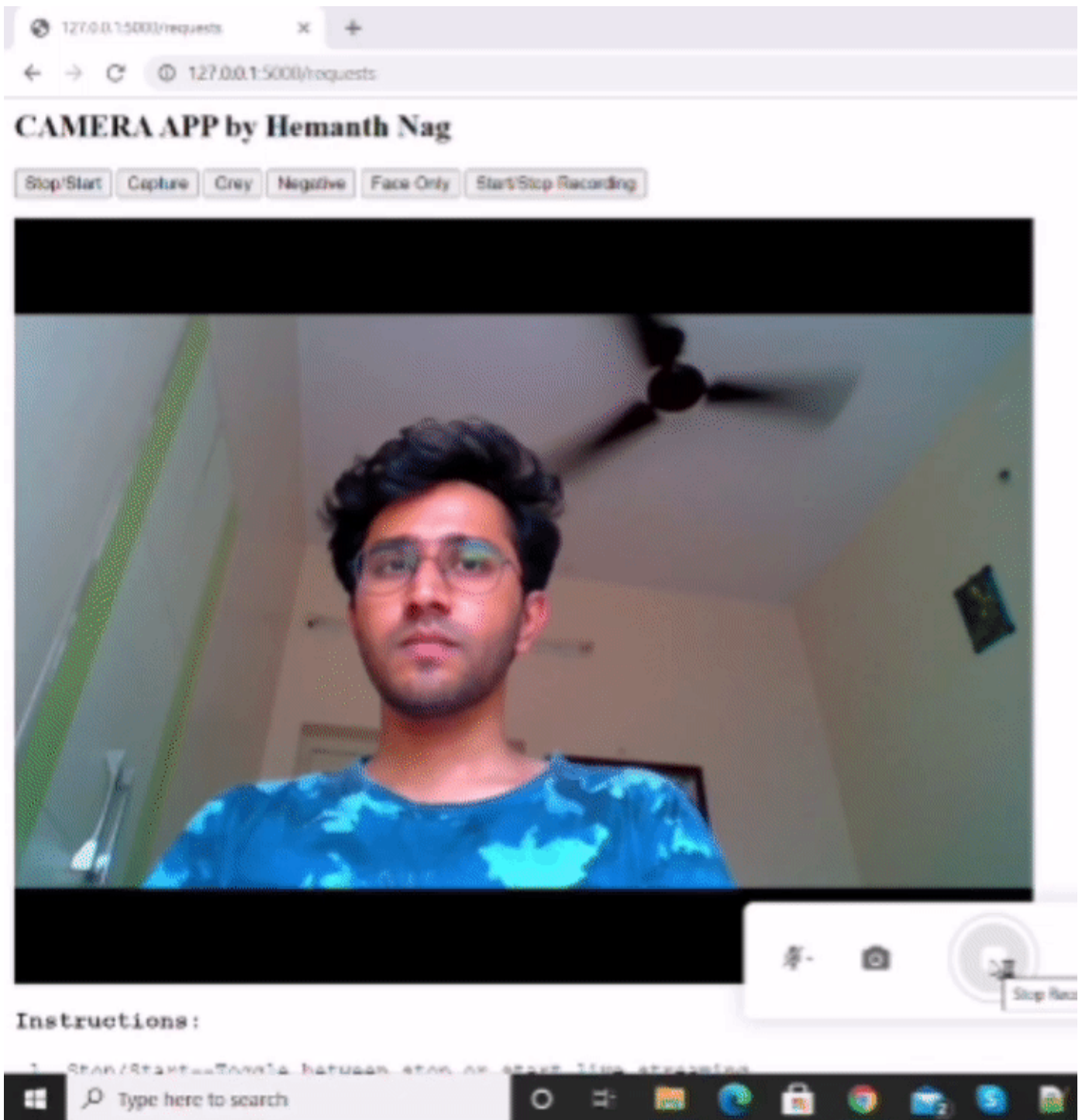
Build a camera app using flask to apply Snapchat like filters, click photos and record videos...

In this blog post, we are going to build a camera app using flask framework wherein we can click pictures, record videos, apply filters like greyscale, negative and 'face only' filter like the one that appears on Snapchat. I have used a very basic design for the front-end since the main motive behind the project was to familiarize myself with the flask web-framework and also include live videostream. The same can be scaled up to add many more features.



Photo by [Bastian Riccardi](#) on [Unsplash](#)

Demo:



GIF by author

We make use of concepts like *threading*, *HTTP request-response*, *global variables*, *error handling* and *face detection*. Let's look at how all these play out in a detailed manner.

Front-end:

Firstly, the front-end is a basic HTML file with buttons to take inputs and image source tag to display output frames after pre-processing in the back-end. Buttons in the file

posts data to the server. The file also displays a few instructions to use the app. This file is saved inside the 'templates' folder in the project directory.

```
1
2  <body>
3  <div class="container">
4      <div class="row">
5          <div class="col-lg-8 offset-lg-2">
6
7
8              <h2 class="mt-5">CAMERA APP by Hemanth Nag</h2>
9                  <form method="post" action="{{ url_for('tasks') }}">
10                      <input type="submit" value="Stop/Start" name="stop" />
11                      <input type="submit" value="Capture" name="click" />
12                      <input type="submit" value="Grey" name="grey" />
13                      <input type="submit" value="Negative" name="neg" />
14                      <input type="submit" value="Face Only" name="face" />
15                      <input type="submit" value="Start/Stop Recording" name="rec" />
16                  </form>
17                  
18                  <h3 style="font-family:courier;">Instructions:</h3>
19                  <ol style="font-family:courier;">
20                      <li>Stop/Start--Toggle between stop or start live streaming</li>
21                      <li>Capture--Take still-shot and save in the 'shots' directory</li>
22                      <li>Grey--Toggle between grayscale and RGB output</li>
23                      <li>Negative--Toggle between negative and RGB output</li>
24                      <li>Face Only--Shows just your face if present(Toggle on/off)</li>
25                      <li>Start/Stop Recording--Toggle between starting and stopping video recor
26
27              </div>
28          </div>
29  </div>
30 </body>
```

CAMERA APP by Hemanth Nag



Instructions :

1. Stop/Start--Toggle between stop or start live streaming
2. Capture--Take still-shot and save in the 'shots' directory
3. Grey--Toggle between grayscale and RGB output
4. Negative--Toggle between negative and RGB output
5. Face Only--Shows just your face if present(Toggle on/off)
6. Start/Stop Recording--Toggle between starting and stopping video recording

Screenshot by author

Back-end:

As for the back-end, it is a single python script which does all the magic. It is saved in the project directory. Let's look at parts of the file separately in order to understand it's working.

Initialization:

```
1  from flask import Flask, render_template, Response, request
2  import cv2
3  import datetime, time
4  import os, sys
5  import numpy as np
6  from threading import Thread
7
8
9  global capture, rec_frame, grey, switch, neg, face, rec, out
10 capture=0
11 grey=0
12 neg=0
13 face=0
14 switch=1
15 rec=0
16
17 #make shots directory to save pics
18 try:
19     os.mkdir('./shots')
20 except OSError as error:
21     pass
22
23 #Load pretrained face detection model
24 net = cv2.dnn.readNetFromCaffe('./saved_model/deploy.prototxt.txt', './saved_model/res10_300x300_s
25
26 #instantiate flask app
27 app = Flask(__name__, template_folder='./templates')
28
29
30 camera = cv2.VideoCapture(0)
```

camera_flask_app.py hosted with ❤ by GitHub

[view raw](#)

In the code above, we import all necessary modules.

Flask is a micro web-framework which works like a bridge between front and back-end. From *flask*, we import ‘*Response*’ and ‘*request*’ modules to handle HTTP response-requests. ‘*render_template*’ is used to render the HTML file shown before. *OpenCV* is the module used to perform all the computer vision tasks. ‘*Thread*’ module is used to spawn new threads.

We then declare all the global variables that act like a ‘toggle switch’ to perform different tasks like capture image, start/stop recording and applying filters. Initialize the variables to 0 to set everything to false.

In line 18, we try to create a folder named ‘shots’ if it doesn’t exist. This is where all the captured images are saved.

Line 24 loads a pretrained face detection model for future uses and line 27 creates an instance of the Flask app. Line 30 creates a video capture object for the built-in camera.

Functions:

```
1  def record(out):
2      global rec_frame
3      while(rec):
4          time.sleep(0.05)
5          out.write(rec_frame)
```

camera_flask_app.py hosted with ❤ by GitHub

[view raw](#)

‘record’ function is used to start recording i.e. write frames into an avi file while ‘rec’ variable is set to true. It uses ‘out’ which is an object for video writer initialized later. (Note: If you feel recorded video is fast or slow, fiddle around with the value of time.sleep).

```

1  def detect_face(frame):
2      global net
3      (h, w) = frame.shape[:2]
4      blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
5          (300, 300), (104.0, 177.0, 123.0))
6      net.setInput(blob)
7      detections = net.forward()
8      confidence = detections[0, 0, 0, 2]
9
10     if confidence < 0.5:
11         return frame
12
13     box = detections[0, 0, 0, 3:7] * np.array([w, h, w, h])
14     (startX, startY, endX, endY) = box.astype("int")
15     try:
16         frame=frame[startY:endY, startX:endX]
17         (h, w) = frame.shape[:2]
18         r = 480 / float(h)
19         dim = ( int(w * r), 480)
20         frame=cv2.resize(frame,dim)
21     except Exception as e:
22         pass
23     return frame

```

camera_flask_app.py hosted with ❤ by GitHub

[view raw](#)

‘*detect_face()*’ takes camera frames as input and returns a cropped out frame containing just the face detected in that frame. It uses the pretrained face detection model loaded earlier.*(Please visit [this website](#) to get an in-depth understanding of how this is being done).*


```

1  def gen_frames(): # generate frame by frame from camera
2      global out, capture, rec_frame
3      while True:
4          success, frame = camera.read()
5          if success:
6              if(face):
7                  frame= detect_face(frame)
8              if(grey):
9                  frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
10             if(neg):
11                 frame=cv2.bitwise_not(frame)
12             if(capture):
13                 capture=0
14                 now = datetime.datetime.now()
15                 p = os.path.sep.join(['shots', "shot_{}.png".format(str(now).replace(":", ''))])
16                 cv2.imwrite(p, frame)
17
18             if(rec):
19                 rec_frame=frame
20                 frame= cv2.putText(cv2.flip(frame,1), "Recording...", (0,25), cv2.FONT_HERSHEY_SIMP
21                 frame=cv2.flip(frame,1)
22
23
24             try:
25                 ret, buffer = cv2.imencode('.jpg', cv2.flip(frame,1))
26                 frame = buffer.tobytes()
27                 yield (b'--frame\r\n'
28                        b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
29             except Exception as e:
30                 pass
31
32         else:
33             pass

```

camera_flask_app.py hosted with ❤ by GitHub

[view raw](#)

‘*gen_frames*’ is an important function wherein the actual frame capture(through the camera) and processing is done. It runs in an infinite while loop. Line 4 captures frames from the camera object. If frame capture is successful, it checks if any of the filter switches are true. If ‘*face*’, ‘*neg*’ or ‘*grey*’ are true, face filter, negative filter and greyscale filter are applied on the read frame respectively.

If *'capture'* variable is set to true, it is reset to false(global variable) and the current frame is saved in *'png'* format. If *'rec'* is true, frame is copied to *'rec_frame'* global variable which is saved into a video file when triggered.

Line 25 encodes the frame into the memory buffer and is then converted into an array of bytes. Line 27 yields the frame data in a format required to be sent as a HTTP response.

HTTP Routes:

This is where the front-end communicates with the server. The communication happens in *'GET'* and *'POST'* methods through URL routes.

```
1 @app.route('/')
2 def index():
3     return render_template('index.html')
```

camera_flask_app.py hosted with ❤ by GitHub

[view raw](#)

'@app. route('/')' is a Python decorator that Flask provides to assign URLs to functions in our app easily. Route *'/'* is the root URL and *'index()'* function is called when root URL is entered. *'index.html'* file is rendered from the function into the webpage.

```
1 @app.route('/video_feed')
2 def video_feed():
3     return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
```

camera_flask_app.py hosted with ❤ by GitHub

[view raw](#)

Route *'/video_feed'* is set as the source for image in the html file. This function returns response chunks of frames yielded by *'gen_frames()'* on a loop.

```

1  @app.route('/requests',methods=['POST','GET'])
2  def tasks():
3      global switch,camera
4      if request.method == 'POST':
5          if request.form.get('click') == 'Capture':
6              global capture
7              capture=1
8          elif request.form.get('grey') == 'Grey':
9              global grey
10             grey=not grey
11         elif request.form.get('neg') == 'Negative':
12             global neg
13             neg=not neg
14         elif request.form.get('face') == 'Face Only':
15             global face
16             face=not face
17             if(face):
18                 time.sleep(4)
19         elif request.form.get('stop') == 'Stop/Start':
20
21             if(switch==1):
22                 switch=0
23                 camera.release()
24                 cv2.destroyAllWindows()
25
26             else:
27                 camera = cv2.VideoCapture(0)
28                 switch=1
29         elif request.form.get('rec') == 'Start/Stop Recording':
30             global rec, out
31             rec= not rec
32             if(rec):
33                 now=datetime.datetime.now()
34                 fourcc = cv2.VideoWriter_fourcc(*'XVID')
35                 out = cv2.VideoWriter('vid_{}.avi'.format(str(now).replace(":",')), fourcc, 20.0,
36                 #Start new thread for recording the video
37                 thread = Thread(target = record, args=[out.1])

```

Open in app 

Get unlimited access



Search Medium



```

42
43     elif request.method=='GET':
44         return render_template('index.html')
45     return render_template('index.html')

```

```
43 return render_template( 'index.html' )
```

camera_flask_app.py hosted with ❤ by GitHub [view raw](#)

information and also sends information. All previous routes were 'GET' by default.

If HTTP method from the client side is 'POST', `request.form.get` accepts data from the buttons pressed by the user and reverses the previous state of the global variables that act like switches in the `gen_frame` function. For example, when user presses 'Grey' button, `grey` global variable is set to True thereby turning the frames to greyscale in the `gen_frames` function. When the 'Grey' button is pressed again, `grey` is set to false turning the frames back to normal.

Recording the frames into a video while also running the flask app is quite tricky. The easiest solution is to start a new *thread*.

*A **thread** is a separate flow of execution. This means that your program will have two things happening at once. A **thread** shares information like data segment, code segment, files etc. with its peer **threads** while it contains its own registers, stack, counter etc.*



In our case, `record()` function has its own while loop, so that loop runs in the new thread. First, we create a `VideoWriter` object when `rec` is true. In line 37, we initialize a new thread with target being `record()` function and line 38 starts the new thread where `record()` function is run. When record button is pressed again, `VideoWriter` object is released and the recording stops to save the video in the root directory.

Finally, if HTTP method from client side is 'GET', `index.html` template is rendered.

Main function:

```
1 if __name__ == '__main__':
2     app.run()
```

camera_flask_app.py hosted with ❤ by GitHub [view raw](#)

 67 |  5 | ...

`app.run()` is used to start the flask app in its default address: <http://127.0.0.1:5000/>. You can set a different host and port number by adding 'host' and 'port' arguments to the function `run`. Setting host to broadcast address `0.0.0.0` would make the app visible in

the whole local area network(wifi, etc). So, you can access the app from your mobile device if it's connected to the same Wi-Fi. This makes a good 'SPY CAM'.

Conclusion:

To run this app, you should have python, flask and OpenCV installed on your PC. To start the app, move to the project directory in the command prompt. Type and enter:

```
python camera_flask_app.py
```

Now, copy-paste <http://127.0.0.1:5000/> into your favorite internet browser and that's it.

You can add more features like AI filters to build a Snapchat-esque app . You can also enhance the user interface making it more interactive and colorful. **You can get the source code for this project in [my GitHub account](#).**

Thank You!!

Flask Framework

Opencv

Videostream

Threading

Image Filter

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Emails will be sent to hector.gonzalezbaron@student.fairfield.edu. [Not you?](#)



Get this newsletter