EmoFilter: una piattafroma per applicare filtri emotion-based

JojoSys

1 Introduzione

EmoFilter è una webapp scritta in python/Flask in grado di applicare in tempo reale un filtro al volto dell'utente sulla base dell'emozione rilevata dal volto dell'utente appoggiandosi ad un sistema esterno di rilevamento delle emozioni. Le emozioni rilevate sono 6:

1. Rabbia
2. Disgusto
3. Paura
4. Felicità
5. Neutralità
6. Tristezza
7. Sorpresa

L'immagine è catturata dallo stream video lato client mediante il browser, i frames contenenti il volto sono inviati al server che gira il frame al sistema di rilevamento delle emozioni per rilevare l'emozione dell'utente nel frame e sulla base dell'emozione rilevata applicare uno dei sette filtri presenti di default.

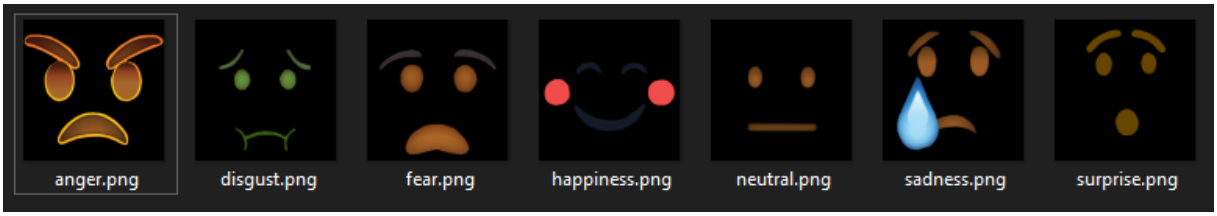

Figure 1: Filtri del sistema

Il frame filtrato è inviato al client ed è visualizzato in tempo reale, questo flusso filtrato può essere catturato dall'utente in una immagine da salvare in locale e/o condividerla mediante Facebook e Instagram, piattaforme social molto utilizzate dalla maggior parte delle persone.

2 Implementazione del server

Il backend,nonché il cuore dell'applicazione, è scritto in Python usando il popolare per lo sviluppo web micro-framework *Flask*, il server acquisisce in tempo reale i frames dal client. Questi frames sono processati dal modulo appositamente creato *get_frames.py*

```python
# generate frame by frame from camera
def gen_frames():
    global frame, frame_not_filtered
    while True:
        # Capture frame-by-frame
        get_frames.lock.acquire()
        frame = get_frames.frame_filtered
        frame_not_filtered = get_frames.frame_not_filtered
        get_frames.lock.release()  # read the camera frame
        if isinstance(frame, ndarray):
            ret, buffer = imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')  # concat frame one by one and show result
```

Figure 2: Funzione principale lato server

Il modulo *get_frames.py* utilizza la libreria **OpenCV** per applicare direttamente sulla porzione del volto dell'utente il filtro selezionato, al momento della selezione del frame ne vengono create due copie:

1. Frame su cui applicare il filtro
2. Frame originale senza filtro

Questo approccio sfrutta il multi-threading per poter condividere i frame tra i vari moduli ed evitare problemi come la *race-condition*.

```python
# used to load the stream video and apply filter on it
def video_stream_filtered():
    offset = []
    global cap, intensity, emotion, frame_filtered, frame_not_filtered
    emotions = filters.Filters()
    emotion_local = None
    ret, frame = cap.read()
    img_h, img_w = frame.shape[:2]
    lock.acquire()
    frame_not_filtered = frame
    lock.release()
    while cap.isOpened() and ret:
        lock.acquire()
        ret, frame = cap.read()
        frame_not_filtered = frame.copy()
        emotion_local = emotion
        intensity_local = intensity
        lock.release()
        if emotion_local != "no_face" and emotion_local != None:
            mask_s_temp, offset = select_emotion(
                emotion_local, emotions, intensity)
            frame = apply_filter(frame, mask_s_temp, offset, img_w, img_h)
        lock.acquire()
        frame_filtered = frame
        lock.release()
        # cv2.imshow('img',frame)
        # cv2.waitKey(1)
    cap.release()
    cv2.destroyAllWindows()
```

Figure 3: Funzione per filtrare lo stream

La funzione acquisisce i possibili filtri mediante la classe *filters* invocando il costruttore, per poi acquisire lo stream video mediante la funzione *cap.read()* sfruttando i lock forniti dai thread. In seguito è creata una copia del frame che permette il salvataggio del volto sul server senza il filtro.
Se un'emozione è rilevata, la funzione applicherà il filtro richiamando la funzione *apply_filter()* per poi aggiornare la risorsa condivisa contente il frame con il filtro applicato.

2.1 load_camera_server()

```python
def load_camera_server():
    global cap, emotion, intensity
    i = 0
    frame = []
    while(True):  # cap.isOpened()
        lock.acquire()
        ret, frame = cap.read()
        lock.release()
        if ret:
            url = 'https://www.intintlab.uniba.it/face-analyzer'
            # encode il numpy array nel formato specificato (jpg), secondo valore e' array monodimensionale numpy dell'immagine
            is_success, im_buf_arr = cv2.imencode(".jpg", frame)
            # converte l'array in byte reali
            byte_frame = im_buf_arr.tobytes()
            # image to send
            my_img = {'image': byte_frame}
            # data to send to server
            my_data = {"emotion": "yes", "gender": "no",
                       "age": "no", "detect_face": "yes"}
            try:
                r = requests.post(url, data=my_data,
                                  files=my_img, verify=False)
```

Questa funzione acquisisce il frame non ancora filtrato, lo codifica in formato jpeg per poi trasformare l'immagine in un flusso di bytes che sarà inviato al sistema **Fce-analyzer** sviluppato dall'intintlab in collaborazione con il dottor. Macchiarulo. Questo flusso di bytes è inviato all'interno di una richiesta in formato json, il sistema fornisce una risposta (sempre in json) contenente informazioni tra cui l'emozione rilevata (se è presente un volto) e l'intensità, parametri usati come già detto per l'applicazione dei filtri.

2.2 Funione apply_filter()

```python
def apply_filter(frame, mask_c_temp, offset, img_w, img_h):
    mask_c = mask_c_temp
    # dimension of the filter
    original_mask_c_h, original_mask_c_w, mask_c_channels = mask_c.shape
    # grey of the filter
    mask_c_grey = cv2.cvtColor(mask_c, cv2.COLOR_BGR2GRAY)
    # mask of the filter
    ret, original_mask_c = cv2.threshold(
        mask_c_grey, 10, 255, cv2.THRESH_BINARY_INV)
    # inverted mask of the filter
    original_mask_inv = cv2.bitwise_not(original_mask_c)
    # gray version of fram
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # faces recognized from the face_cascade
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

```
if len(faces) != 0:
    # coordinates of face region
    face_w = faces[0][2]  # w
    face_h = faces[0][3]  # h
    face_x1 = faces[0][0]  # x
    face_x2 = face_x1 + face_w
    face_y1 = faces[0][1]  # y
    face_y2 = face_y1 + face_h

    # filter size in relation to face by scaling
    mask_c_width = int(offset[0] * face_w)
    mask_c_height = int(
        mask_c_width * original_mask_c_h / original_mask_c_w)

    # setting location of coordinates of witch
    mask_c_x1 = face_x2 - int(face_w/2) - int(mask_c_width/2) + offset[1]
    mask_c_x2 = mask_c_x1 + mask_c_width
    mask_c_y1 = face_y1 + offset[2]
    mask_c_y2 = mask_c_y1 + mask_c_height
```

```
    # check to see if out of frame
    if mask_c_x1 < 0:
        mask_c_x1 = 0
    if mask_c_y1 < 0:
        mask_c_y1 = 0
    if mask_c_x2 > img_w:
        mask_c_x2 = img_w
    if mask_c_y2 > img_h:
        mask_c_y2 = img_h

    # account for any out of frame changes
    mask_c_width = mask_c_x2 - mask_c_x1
    mask_c_height = mask_c_y2 - mask_c_y1
    # resize witch to fit on face
```

```
    try:
        mask_c = cv2.resize(
            mask_c, (mask_c_width, mask_c_height), interpolation=cv2.INTER_AREA)
        mask = cv2.resize(original_mask_c, (mask_c_width,
                                            mask_c_height), interpolation=cv2.INTER_AREA)
        mask_inv = cv2.resize(
            original_mask_inv, (mask_c_width, mask_c_height), interpolation=cv2.INTER_AREA)
        #entered = True

        # take roi(region of interest) for filter from background
        roi = frame[mask_c_y1:mask_c_y2, mask_c_x1:mask_c_x2]

        # changing color pixel in the frame and in the filter and adding the filter on the frame
        roi_bg = cv2.bitwise_and(roi, roi, mask=mask)
        roi_fg = cv2.bitwise_and(mask_c, mask_c, mask=mask_inv)
        dst = cv2.add(roi_bg, roi_fg)

        # put back in original frame
        frame[mask_c_y1:mask_c_y2, mask_c_x1:mask_c_x2] = dst
    except cv2.error as e:
        pass

return frame
```

Questa funzione riceve in input il frame, la maschera che rappresenta il filtro, un offset array che contiene gli offset per posizionare il filtro sul frame, altezza e larghezza del frame. Come prima cosa viene creata una maschera per il filtro, in seguito viene utilizzato l'algoritmo *haarcascades* fornito dalle librerie Opencv2 per riconoscere le coordinate su cui applicare il filtro.
Se il riconoscimento va a buon fine, il filtro viene applicato sfruttando le *bitwise operation*, per approfondire seguire questo link Documentazione cv2. Dopo aver applicato il filtro, la funzione restituisce il frame modificato.

```
def select_emotion(emotion_local,emotions,intensity):
    dim = []
    if emotion_local == "Anger":
        if intensity > 5:
            mask_s_temp = emotions.veryAnger[0]
            dim.append(emotions.veryAnger[1])
            dim.append(emotions.veryAnger[2])
            dim.append(emotions.veryAnger[3])
        else:
```

Questa funzione seleziona il filtro da applicare in base all'emozione rilevata

3 Classe Filters

```python
class Filters:
    def __init__(self):
        self.anger = [cv2.imread('filters_folder/filters_img/anger.png'),1,0,0]
        self.veryAnger = [cv2.imread('filters_folder/filters_img/veryAnger.png'),1,0,20]
        self.disgust = [cv2.imread('filters_folder/filters_img/disgust.png'),1,0,0]
        self.veryDisgust = [cv2.imread('filters_folder/filters_img/veryDisgust.png'),1,0,25]
        self.fear = [cv2.imread('filters_folder/filters_img/fear.png'),1,0,0]
        self.veryFear = [cv2.imread('filters_folder/filters_img/veryFear.png'),1,0,20]
        self.happiness = [cv2.imread('filters_folder/filters_img/happiness.png'),1,0,20]
        self.veryHappiness = [cv2.imread('filters_folder/filters_img/veryHappiness.png'),1,0,20]
        self.neutral = [cv2.imread('filters_folder/filters_img/neutral.png'),1,0,5]
        self.veryNeutral = [cv2.imread('filters_folder/filters_img/veryNeutral.png'),1,0,20]
        self.sadness = [cv2.imread('filters_folder/filters_img/sadness.png'),1,0,20]
        self.verySadness = [cv2.imread('filters_folder/filters_img/verySadness.png'),1,0,10]
        self.surprise = [cv2.imread('filters_folder/filters_img/surprise.png'),1,0,20]
        self.verySurprise = [cv2.imread('filters_folder/filters_img/verySurprise.png'),1,0,20]

class Filter:
    def __init__(self,path,offset_w,offset_y1):
        self.path = path
        self.offset_w = offset_w
        self.offset_y1 = offset_y1
```

Questa classe è utilizzata come database per caricare risorse quali:

- immagini
- scale del filtro
- offset x ed y per posizionare il filtro

Una volta caricata la classe **Filters** sono disponibili tutte le risorse per l'applicazione dei filtri. A livello pratico questo approccio risulta molto efficiente sia durante l'esecuzione del codice che per eventuali modifiche alle immagini dei filtri applicati.