TP2

January 4, 2024

Sébastien BOIS & Maxime BOUET

1 Snippet to choose the right frames

1.1 Filter using HSV and RGBA space colors

Here, the function is used to create a mask fitting to the color of the skin. In this case, we consider two color spaces to filter the skin color: HSV and RGBA. The algorithm is the following: * In theory for the HSV space color, we use the following values: 0 <= H <= 128, 58 <= S <= 174, 0 <= V <= 255. However, due to a bad quality of artificial lighting and of webcam, we made the choice to allow a bigger saturation range going from 0 to 174. * Concerning the RGBA space color, the values are the followings: R > 95, G > 40, B > 20 and A > 15. There also are some conditions between each values: R > G, R > B and |R-G| > 15

Combining all these conditions between the two different spaces allows to detect the skin color effectively.

[1]:
```
def custom_mask(frame):
    """
    Skin detection mask using RGBA and HSV spaces
    Combines several mask to obtain the better accuracy on the skin color␣
 ↪detection

    returns the cut out skin.
    """
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    rgba = cv.cvtColor(frame, cv.COLOR_BGR2RGBA)

    # For the saturation values, it should be 0.23 <= S <= 0.68 <=> 58 <= S <=␣
 ↪174
    # However, because of the ambiant light and of the quality of the camera,␣
 ↪we had to
    # adapt it due to some important saturation.
    hsv_lower = np.array([0,0,0])
    hsv_upper = np.array([128,174,255])

    rgba_lower = np.array([95,40,20,15])
    rgba_upper = np.array([255,255,255,255])
```

```
    hsv_mask = cv.inRange(hsv, hsv_lower, hsv_upper)
    rgba_mask = cv.inRange(rgba, rgba_lower, rgba_upper)
    diff_mask = (np.logical_and(rgba[:,:,0]>rgba[:,:,1], np.logical_and(rgba[:,:
↪,0]>rgba[:,:,2], np.abs(rgba[:,:,0]-rgba[:,:,1])>15))*255).astype(np.uint8)

    comb_mask = cv.bitwise_and(diff_mask,rgba_mask)
    mask = cv.bitwise_and(hsv_mask,comb_mask)
    return mask
```

1.2 Filter using RGBA and YcrCb space colors

To try out the effectiveness of algorithms, we implemented a second one using the RGBA and YCrCB space colors. The algorithm is the following: * For the RGBA space color, the values are the following: R > 95, G > 40, B > 20 and A > 15. There also are the same conditions between the fields as the previous filter: R > G, R > B and |R-G| > 15. * For the YCrCb space color, the fields must fulfill these conditions: Y > 80, Cr > 135, Cb > 85, Cr <= (1.5862 * Cb + 20), Cr >= (0.3448 * Cb + 76.2029), Cr >= (-4.5652 * Cb + 234.5652), Cr <= (-1.15 * Cb + 301.75) and Cr <= (-2.2857 * Cb + 432.85).

Again, we combine the conditions between the two color spaces to create a complete and complex mask cutting the skin out of the original image.

[2]:
```
def custom_mask2(frame):
    """
    Skin detection mask using RGBA and YCrCb space colors.
    Combines several mask to obtain the better accuracy on the skin color␣
↪detection

    returns the cut out skin.
    """
    rgba = cv.cvtColor(frame, cv.COLOR_BGR2RGBA)
    ycrcb = cv.cvtColor(frame,cv.COLOR_BGR2YCR_CB)

    rgba_lower = np.array([95,40,20,15])
    rgba_upper = np.array([255,255,255,255])

    ycrcb_lower = np.array([80, 135, 85])
    ycrcb_upper = np.array([255, 255, 255])

    rgba_mask = cv.inRange(rgba, rgba_lower, rgba_upper)
    ycrcb_mask = cv.inRange(ycrcb, ycrcb_lower, ycrcb_upper)

    # Conditions on the RGBA fields
    diff_mask1 = (np.logical_and(
                    rgba[:,:,0]>rgba[:,:,1],
                    np.logical_and(
                        rgba[:,:,0]>rgba[:,:,2],
```

```
                        np.abs(rgba[:,:,0]-rgba[:,:,1])>15
                    )
                )*255).astype(np.uint8)

    # Conditions on the YCrCb fields
    diff_mask2 = (np.logical_and(
        np.logical_and(
            np.logical_and(
                np.logical_and(
                    ycrcb[:,:,1] >= 0.3448 * ycrcb[:,:,2] + 76.2069,
                    ycrcb[:,:,1] <= 1.5862 * ycrcb[:,:,2] + 20),
                ycrcb[:,:,1] >= -4.5652 * ycrcb[:,:,2] + 234.5652),
            ycrcb[:,:,1] <= -1.15 * ycrcb[:,:,2] + 301.75),
        ycrcb[:,:,1] <= -2.2857 * ycrcb[:,:,2] + 432.85)).astype(np.uint8)

    comb_mask1 = cv.bitwise_and(rgba_mask, diff_mask1)
    comb_mask2 = cv.bitwise_and(ycrcb_mask, diff_mask2)
    mask = cv.bitwise_and(comb_mask1, comb_mask2)
    return mask
```

1.3 Capture images with the skin detection masks

This script has been written from the OpenCV documentation and slightly modified to fit to our task. It captures the videostream from a specific webcam (When a webcam is natively included in the laptop, the right VideoCapture entry is 0). It displays three images: the raw image, the raw image with the first skin color filter applied and the raw image with the second skin color filter applied. When the result satisfies you, you must press the key 's' in order to save the pictures.

[4]:
```
import cv2 as cv
import numpy as np

# Captures the video stream from the default webcam device
cap = cv.VideoCapture(2) #2
# If the capture works
if cap.isOpened():
    while True:
        ret, frame = cap.read() # Get frames one by one
        if not ret: # If the return of the read is False, it means there is a␣
↪problem
            break
        # Our operations on the frame come here
        mask = custom_mask(frame)
        mask2 = custom_mask2(frame)

        res = cv.bitwise_and(frame,frame,mask=mask)
        res2 = cv.bitwise_and(frame,frame,mask=mask2)
```

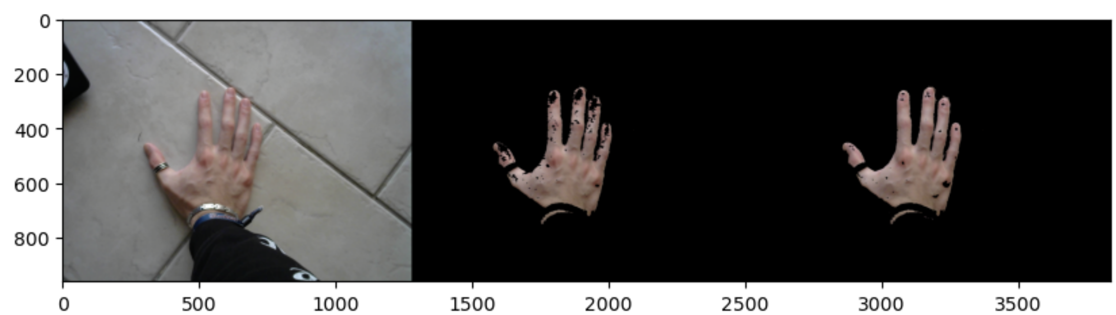
```
        render = np.concatenate((frame,res,res2),axis=1)
        # Display the resulting frame
        cv.imshow('frame1', render)
        if cv.waitKey(1) == ord('s'):
            model = frame
            final = res
            final2 = res2
            break

cap.release()
cv.destroyAllWindows()
```

```
[ WARN:0@24.301] global ./modules/videoio/src/cap_gstreamer.cpp (1405) open
OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1,
duration=-1
```

Lastly, when the images fits to the criterias, we can run the cell below to save the results into different files at the root of the project. It creates three different images: * The skin cut out with the first mask (HSV and RGBA) * The skin cut out with the second mask (YCrCb and RGBA) * The raw image that will be manually cut out to create the ground truth image

[5]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.imshow(cv.cvtColor(render, cv.COLOR_BGR2RGB))
cv.imwrite('natural_hsv.jpg', final)
cv.imwrite('natural_ycrcb.jpg', final2)
cv.imwrite('natural_model.jpg', model)
```

[5]: True



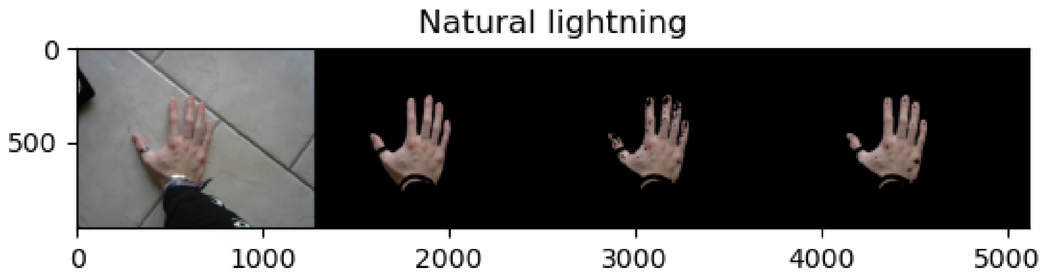
2 Experiments

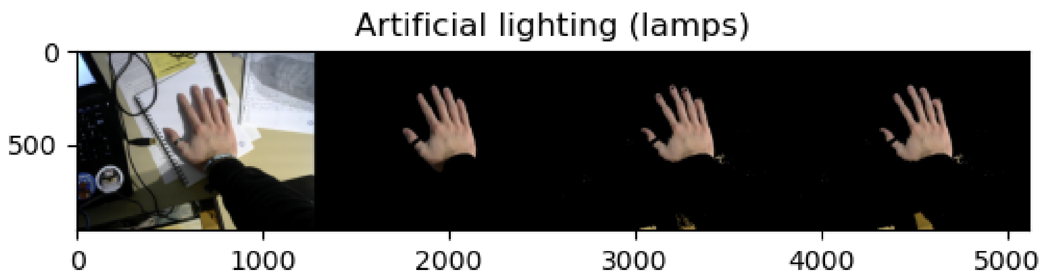
2.1 Loading images from the disk

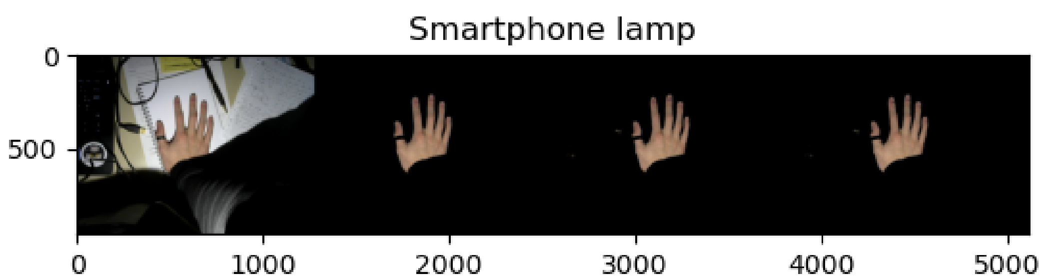
Firstly, we load all the saved images from the disk. It includes model, HSV filtered, YCrCb filtered and ground truth images for the three lighting conditions: natural, artificial and smartphone lamp.

The ground truth images has been manually edited with GIMP to only keep the skin parts. We display the images to spot the differences between each of them.

[13]:
```
# natural light
natural_model = cv.imread('natural_model.jpg')
natural_hsv = cv.imread('natural_hsv.jpg')
natural_ycrcb = cv.imread('natural_ycrcb.jpg')
natural_gt = cv.imread('natural_ground_truth.jpg')
natural = np.concatenate((natural_model, natural_gt, natural_hsv,␣
↪natural_ycrcb), axis=1)
naturals = ['Natural lights', natural_gt, natural_hsv, natural_ycrcb]
# lighting
lighting_model = cv.imread('light_model.jpg')
lighting_hsv = cv.imread('light_hsv.jpg')
lighting_ycrcb = cv.imread('light_ycrcb.jpg')
lighting_gt = cv.imread('light_ground_truth.jpg')
lighting = np.concatenate((lighting_model, lighting_gt, lighting_hsv,␣
↪lighting_ycrcb), axis=1)
lightings = ['Artificial lights', lighting_gt, lighting_hsv, lighting_ycrcb]

# smartphone light
smartphone_model = cv.imread('smartphone_model.jpg')
smartphone_hsv = cv.imread('smartphone_hsv.jpg')
smartphone_ycrcb = cv.imread('smartphone_ycrcb.jpg')
smartphone_gt = cv.imread('smartphone_ground_truth.jpg')
smartphone = np.
↪concatenate((smartphone_model,smartphone_gt,smartphone_hsv,smartphone_ycrcb),␣
↪axis=1)
smartphones = ['Smartphone lamp', smartphone_gt, smartphone_hsv,␣
↪smartphone_ycrcb]

samples = [naturals, lightings, smartphones]

plt.imshow(cv.cvtColor(natural, cv.COLOR_BGR2RGB))
plt.title('Natural lightning')
plt.show()
plt.imshow(cv.cvtColor(lighting, cv.COLOR_BGR2RGB))
plt.title('Artificial lighting (lamps)')
plt.show()
plt.imshow(cv.cvtColor(smartphone, cv.COLOR_BGR2RGB))
plt.title('Smartphone lamp')
plt.show()
```







2.2 Scoring functions

To assess a score to the effectiveness of the algorithms, we must define a scoring function. To do so, we compare the output image to the ground truth image and look for three values: the true positives, false positives and false negatives. From these three statistics, we can obtain two score: * the accuracy: True positives / (True positives + False positives) * the recall: True positives / (True positives + False negatives)

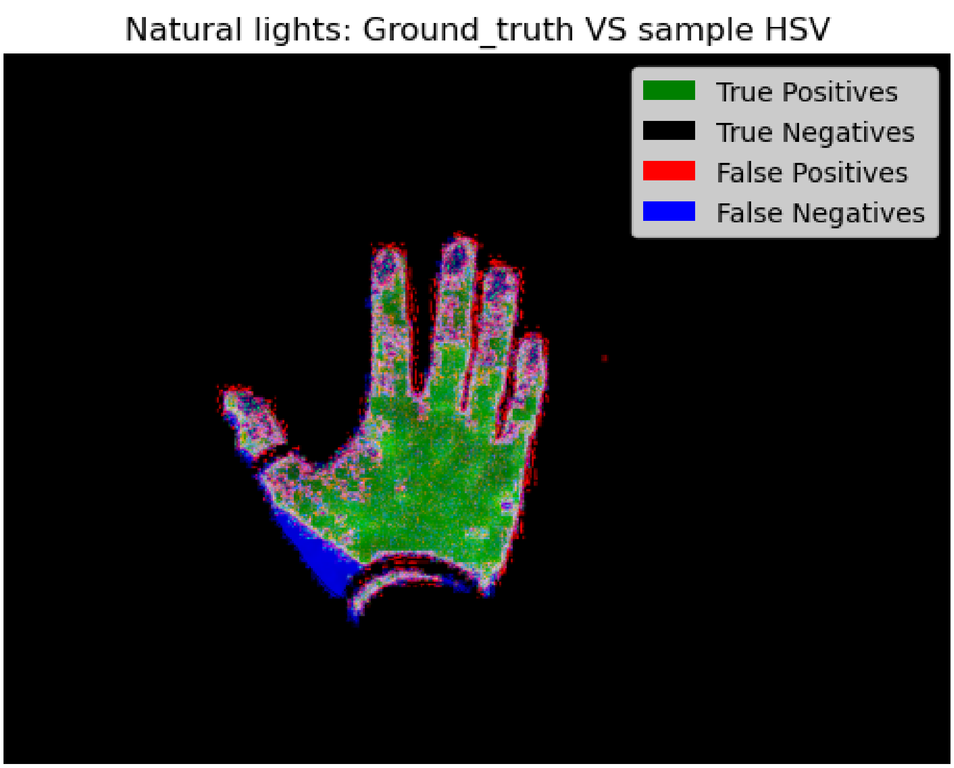
We also return an image highlighting each of the three categories: in green, the true positives, in red, the false positives and in blue, the false negatives. By default, the true negatives are still in black.

[10]:
```python
def set_color(image, rgb):
    """
    Change the color of the pixels of the image by multiplying it pixelwise
    """
    for i in range(len(rgb)):
        image[:,:,i] *= rgb[i]
    return image

def score_img(ground_truth, target):
    """
    Compute the true positives, false positives and false negatives of two␣
 ↪images
    Renders true positives in green, false positives in red and false negatives␣
 ↪in blue
    Counts the number of element for each class and the accuracy and recall␣
 ↪values

    Returns the resulting image and all the statistics computed
    """
    not_gt = cv.bitwise_not(ground_truth)
    not_target = cv.bitwise_not(target)

    tp = set_color(cv.bitwise_and(ground_truth, target), [0,255,0])
    fp = set_color(cv.bitwise_and(not_gt, target), [255, 0, 0])
    fn = set_color(cv.bitwise_and(ground_truth, not_target), [0,0,255])

    plot = cv.bitwise_or(cv.bitwise_or(fp, fn), tp)

    count_tp, count_fp, count_fn = cv.countNonZero(cv.cvtColor(tp, cv.
 ↪COLOR_BGR2GRAY)), cv.countNonZero(cv.cvtColor(fp, cv.COLOR_BGR2GRAY)), cv.
 ↪countNonZero(cv.cvtColor(fn, cv.COLOR_BGR2GRAY))
    accuracy = count_tp/(count_tp+count_fp)
    recall = count_tp/(count_tp+count_fn)

    return plot, count_tp, count_fp, count_fn, accuracy, recall
```

The cell below execute the scoring functions for each sample. It compares both the HSV and YCrCb filtered images to the corresponding ground truth of the sample and plot the results.

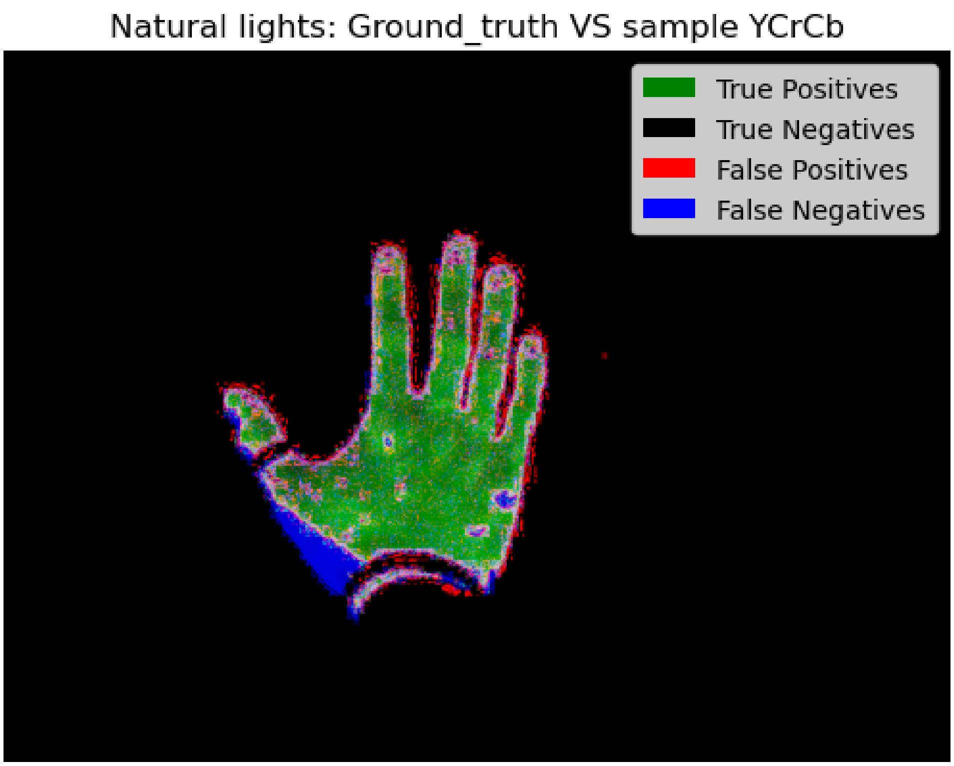
[14]:
```python
title_labels = ['HSV', 'YCrCb']

for samp in samples:
    title = samp[0] # Name of the current light sample
    gt = samp[1] # Ground truth image for the current light sample
    for index in range(2, len(samp)):
        plot, tp, fp, fn, accuracy, recall = score_img(cv.cvtColor(gt, cv.
 ↪COLOR_BGR2RGB), cv.cvtColor(samp[index], cv.COLOR_BGR2RGB))
```

```
        total = len(plot.flatten())
        tn = total - tp - fp - fn
        plt.imshow(plot)

        colors = ['green', 'black', 'red', 'blue']
        labels = ['True Positives', 'True Negatives', 'False Positives', 'False␣
↪Negatives']

        for color, label in zip(colors, labels):
            plt.bar(0, 0, color=color, label=label)

        plt.title(f"{title}: Ground_truth VS sample {title_labels[index-2]}")
        plt.legend()
        plt.axis('off')
        plt.show()
        print(f"True positives: {tp}\nTrue negatives: {tn}\nFalse positives:␣
↪{fp}\nFalse negatives: {fn}\nTotal: {total}\n\nAccuracy: {(accuracy*100):.
↪2f}%\nRecall: {(recall*100):.
↪2f}%\n------------------------------------------------------------")
```

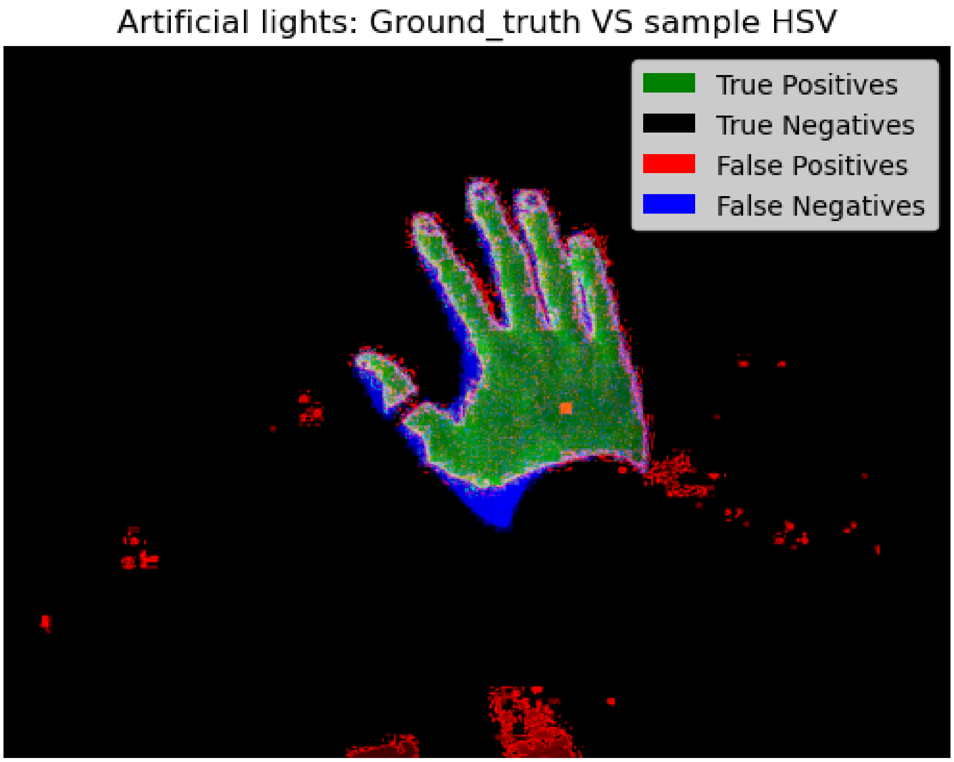



```
True positives: 88113
True negatives: 3503722
```

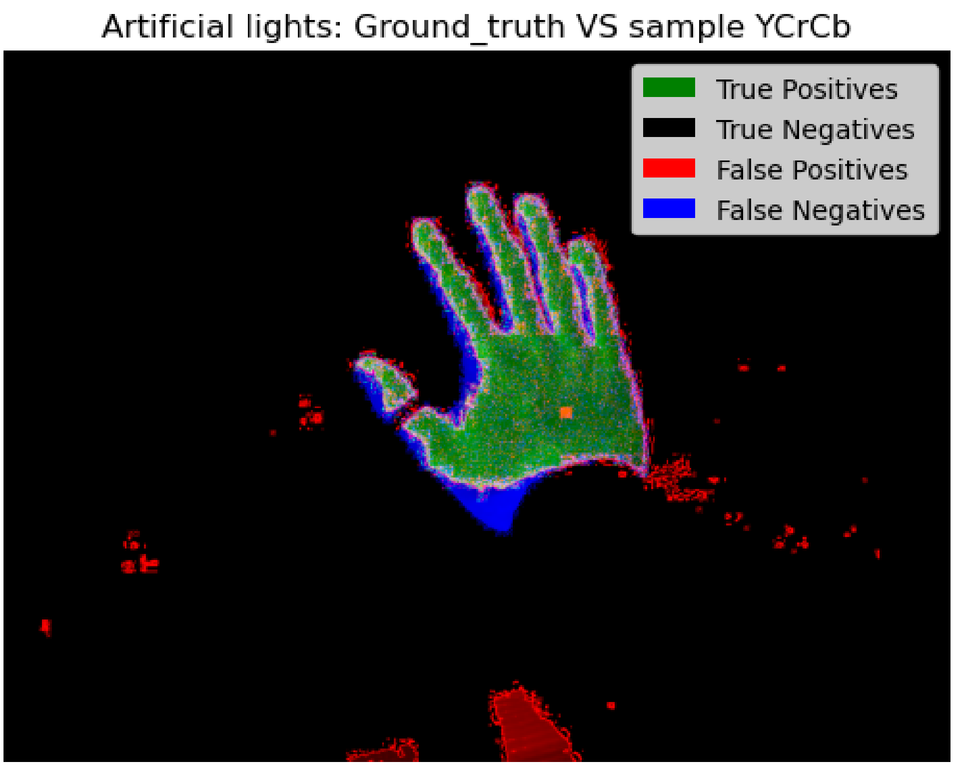
```
False positives: 46066
False negatives: 48499
Total: 3686400

Accuracy: 65.67%
Recall: 64.50%
-----------------------------------------------------------
```



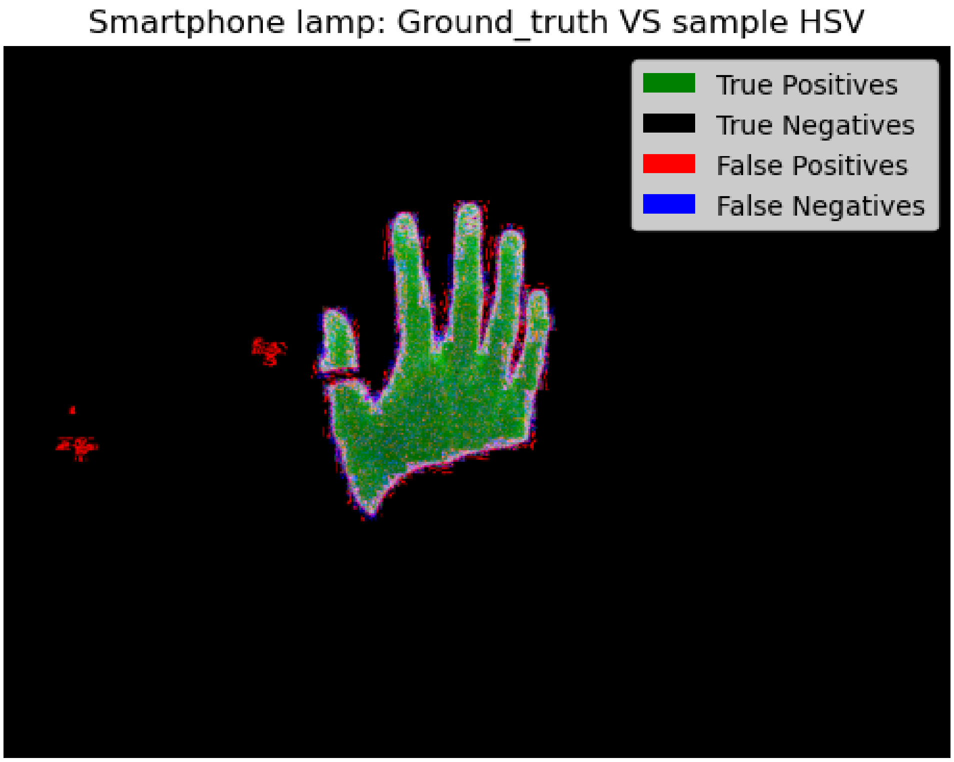

```
True positives: 93705
True negatives: 3512928
False positives: 40827
False negatives: 38940
Total: 3686400

Accuracy: 69.65%
Recall: 70.64%
-----------------------------------------------------------
```



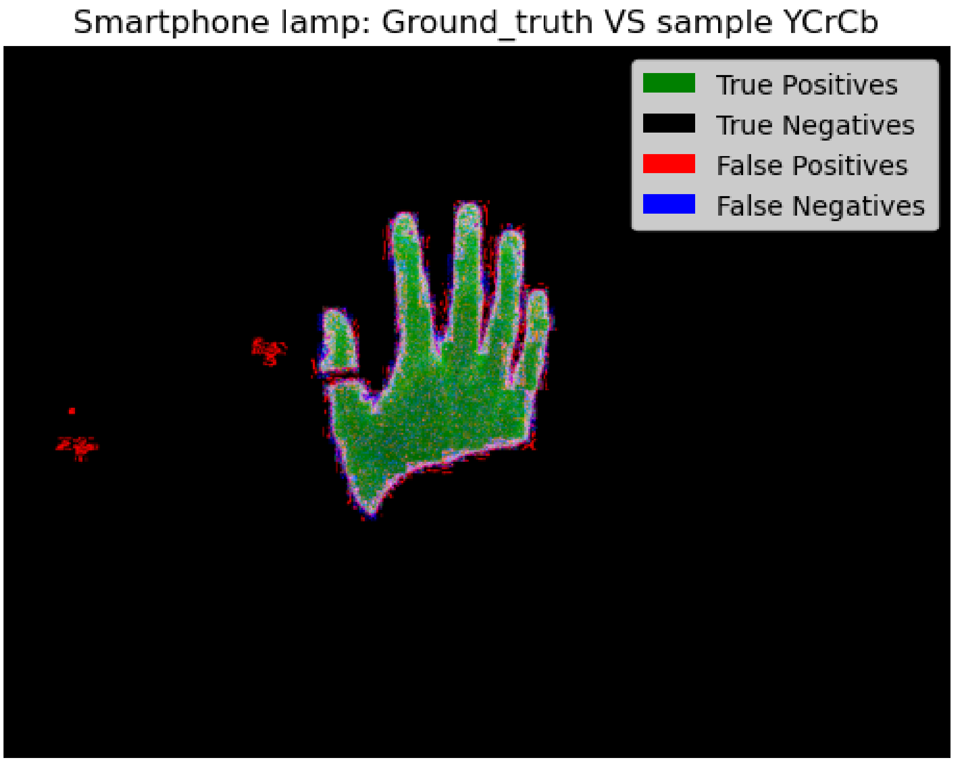

```
True positives: 78441
True negatives: 3525681
False positives: 43435
False negatives: 38843
Total: 3686400

Accuracy: 64.36%
Recall: 66.88%
-----------------------------------------------------------
```




```
True positives: 76891
True negatives: 3524448
False positives: 44768
False negatives: 40293
Total: 3686400

Accuracy: 63.20%
Recall: 65.62%
-----------------------------------------------------------
```




```
True positives: 64715
True negatives: 3568501
False positives: 26888
False negatives: 26296
Total: 3686400

Accuracy: 70.65%
Recall: 71.11%
------------------------------------------------------------
```




```
True positives: 64494
True negatives: 3568235
False positives: 27189
False negatives: 26482
Total: 3686400

Accuracy: 70.34%
Recall: 70.89%
------------------------------------------------------------
```

3 Results

3.1 Impact of the space color

By looking at the different results, we notice that the second filter using both RGBA and YCrCb spaces is slightly better than the first using RGBA and HSV spaces. However, even if it has a small impact on the effectiveness, it does not change drastically the accuracy of the skin detection algorithm.

3.2 Impact of the light quality

The light however has a bigger impact. Indeed, we notice that the image with the smartphone lighting achieves better results than the two others. Indeed, we obtain a 70-71% accuracy and recall on the skin detection for the smartphone lights VS 60-65% for the others. Only the YCrCb filtered image for the sample with natural lighting can match these scores. The light has an impact on the color of the skin, it can make the camera saturate but also create shadows on the skin.

3.3 Other quality factors

There are also other factors affecting the scores of the algorithm. First, the ground truth are not perfect. They were manually created using the free selection tool of GIMP. There can be some mistakes in the kept pixels. Some may have been discarded whereas some others may have been kept when they should not. The angle of the camera to the light also affect a lot the shadows of the pictures and thus the effectiveness of the filters. Lastly, there can also be some objects in the pictures matching the filter conditions but which are not skin. For example, I have a wood table which has often been recognized as skin even if it is not.

3.4 Conclusion

To conclude, the use of the filters helps to detect skin pixels but is not perfect. Indeed, 60-70% accuracy is not a relevant score. It can be used in some predefined conditions but not in real-time. In real conditions, we can't afford to have such vagueness on the detection otherwise it can result with very bad consequences (in automated vehicules for example). We can't always provide a good lighting, with the perfect angle between the ligt and the camera.