Image Processing in Practice — Practice Questions

OpenCV Short Questions

You have created a new C++ project. What are the main steps you need to take to use the OpenCV library from your project?

Select three options from the list:

- I need to add the following line to my source code: `using namespace cv;`
- **I need to specify the path of the OpenCV include files.**
- **I need to specify the name (including the exact version) of the OpenCV library.**
- **I need to specify the path of the OpenCV library (`.lib`) files.**

After executing the following opencv instructions, what can be said about the img matrix?

```
Mat img = Mat::zeros(30, 50, CV_8UC1);
Mat img2 = img.clone();
img2.at<uchar>(0, 0) = 100;
```

Select two options.

- The clone function creates a shallow-copy of the img, so the pixel value in the upper left corner of the `img` will be 100.
- The clone function creates a shallow-copy of the img, so the pixel value in the upper right corner of the `img` will be 100.
- **The clone function creates a deep copy of the img, therefore the modification of `img2` has no effect on it.**
- **The img is a black image.**

After executing the following opencv instructions, what can be said about the result matrix?

```
Mat img(2, 3, CV_8UC3);
Mat img2 = Mat::zeros(2, 3, CV_8UC1);
Mat result = img + img2;
```

Select one:

- **The matrix will not be created because the pixel-wise addition can not be performed between a single-channel and a three-channel image.**
- The result will be a white image with 2 rows and 3 columns.
- The result will be a black image with 3 rows and 2 columns.
- The result will be a white image with 3 rows and 2 columns.

After executing the following OpenCV instruction, what can be said about the img?

```
Mat img(20, 30, CV_8UC1);
```

Select two options:

- **The value of the pixels is not defined.**
- **It is an image with 20 rows and 30 columns.**
- It is a black image.
- The `CV_8UC1` is invalid type, so the matrix will not be created.

We have created an image with the following command:

```
uchar t[] = {255, 2, 0};
Mat img( 1, 3, CV_8U, t);
Mat result = img - 1;
```

Select the one true statement:

- The code causes an exception because the - operation is not interpreted between scalar and matrix.
- **The result matrix contains three values: `255, 1, 0`.**
- The result matrix contains three values: `255, 1, -1`
- The result matrix contains two values: `255, 2`.

Color Models

Color the sun red

1. Read the sun_dog.png in color.
2. The orange sun behind the dog fits into the design, change it to red (red: 255, green: 0, blue: 0). Save the result: result_stud.png

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){
    Mat img = imread("sun_dog.png", IMREAD_COLOR);
    Mat result = img.clone();
    Mat mask;
    Vec3b bgr;
    bgr = img.at<Vec3b>((img.rows / 2), (img.cols / 2));

    inRange(img, Scalar(bgr[0] - 50, bgr[1] - 50, bgr[2] - 50),
        Scalar(bgr[0] + 50, bgr[1] + 50, bgr[2] + 50), mask);
    result.setTo(Scalar(0, 0, 255), mask);

    imwrite("result_stud.png", result);
    return 0;
}
```

Create a Milka cow: recolor the brown pixels to purple.

1. Read the cow_bw.jpg in color.
2. Convert the image to the HSV color space. Save the hsv image: hsv_stud.png
3. Iterate over the hsv image: If a pixel is brown, change its hue component to 140. Save the result: result_hsv_stud.png
4. Convert the modified hsv image back to RGB (BGR in OpenCV) color space. Save the result: result_stud.png

Hint: Brown is dark orange, so just select those points whose hue component is less than 30.

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){

    Mat bgr = imread("cow_bw.jpg", IMREAD_COLOR);
    Mat hsv;
    cvtColor(bgr, hsv, ColorConversionCodes::COLOR_BGR2HSV);
    imwrite("hsv_stud.png", hsv);

    Vec3b hsv_c;
    for (int i = 0; i < hsv.rows; i++) {
        for (int j = 0; j < hsv.cols; j++) {
            hsv_c = hsv.at<Vec3b>(i, j);
            if (hsv_c[0] < 30) {
                hsv_c[0] = 140;
                hsv.at<Vec3b>(i, j) = hsv_c;
            }
        }
    }
    imwrite("result_hsv_stud.png", hsv);

    Mat bgr_modified;
    cvtColor(hsv, bgr_modified, ColorConversionCodes::COLOR_HSV2BGR);
    imwrite("result_stud.png", bgr_modified);

    return 0;
}
```

Copy Fish to Sea

1. Read the fish3.jpg in color.
2. Read the sea_bg.jpg in color.
3. Copy to a fish to the sea image. (The two images are the same size, no need to resize them.) Save the result: result_stud.png

Hint: The blue component is dominant.

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){
    Mat fish = imread("fish3.jpg", IMREAD_COLOR);
    Mat sea = imread("sea_bg.jpg", IMREAD_COLOR);
    Vec3b c_bgr;


    for (int i = 0; i < fish.rows; i++) {
        for (int j = 0; j < fish.cols; j++) {
            c_bgr = fish.at<Vec3b>(i, j);
            if (c_bgr[0] > 0 && c_bgr[1] < 150 && c_bgr[2] < 200) {
                sea.at<Vec3b>(i, j) = c_bgr;
            }
        }
    }

    imwrite("result_stud.png", sea);

    return 0;
}
```

Threshold

Mask the Cat

1. Read the cart_cat.jpg image in grayscale.
2. Use a binary threshold method to get the mask of the cat (The cat pixels should be white on the result image, the background should be black.) Save the mask image: result_stud.png

If the eyes of the cat turn black, it's okay.

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){

    Mat img = imread("cart_cat.jpg", IMREAD_GRAYSCALE);
    Mat result;
    threshold(img, result, 20, 255, THRESH_BINARY);

    imwrite("result_stud.png", result);

    return 0;
}
```

Copy the Cat

1. Read the cart_cat.jpg image in grayscale.
2. Use a special global threshold method. The result image should contain the pixels of the cat (with the original intensity values), the other points should be black.
   Save the result: result_stud.png

If the eyes of the cat turn black, it's okay.

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){

    Mat img = imread("cart_cat.jpg", IMREAD_GRAYSCALE);
    Mat mask;
    threshold(img, mask, 20, 255, THRESH_BINARY);

    Mat result = Mat::zeros(img.size(), CV_8UC3);
    img.copyTo(result, mask);

    imwrite("result_stud.png", result);
    return 0;
}
```

Count dark pixels

1. Read the flower.jpg in grayscale.
2. Darken the image: subtract 50 from the intensity values. Save the darken version: dark_stud.png
3. Count the pixel with 0 intensity value on the darkened image. Print the result as an integer (to the standard output).

Hint for step 3: You can use a for loop to count the black pixels, or a countNonZero function to count the non-black pixels. You can get the size of the image, and calculate the number of the black pixels.

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){

    Mat img = imread("flower.jpg", IMREAD_GRAYSCALE);
    img -= 50;

    imwrite("dark_stud.png", img);

    int pixels = 0;

    for (int i = 0; i < img.rows; i++) {
        for (int j = 0; j < img.cols; j++) {
            if (img.at<uchar>(i, j) == 0) {
                pixels++;
            }
        }
    }

    cout << pixels;

    return 0;
}
```

Mask the cookie

1. Open the vanilia_cookie.jpg image (in grayscale)
2. Resize the image. The expected size of the new image is (100, 100). Save the resized image: resized_stud.png
3. Use a threshold method on the resized version of the image. The cookie should be white, the background should be black in the result image. Save the result: result_stud.png

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){

    Mat img = imread("vanilia_cookie.jpg", IMREAD_GRAYSCALE);
    resize(img, img, Size(100, 100));
    imwrite("resized_stud.png", img);
    imshow("resized", img);

    Mat mask;
    threshold(img, mask, 220, 255, THRESH_BINARY_INV);

    imwrite("result_stud.png", mask);

    return 0;
}
```

Apply the Mask on the cookie.

1. Open the vanilia_cookie.jpg image in grayscale.
2. Resize the image. The width and height of the resized image should be half of the original. The number of the rows is odd, make sure to truncate the fraction part. (Meaning: use integer as a divisor and you will be fine. If you use float, do not round the result up.) Save the resized image: resized_stud.png
3. Use a threshold method on the resized version of the image to get the mask of the cookie. The cookie should be Save the result: mask_stud.png
4. Copy the cookie from the resized image (result of step2) to a new black image. The size and the type of the black image is the same as the resized image's size, but you can let the OpenCV handle it, if you know the easiest way to solve this step. Save the result: result_stud.png

Solution:

```
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;

int main(){

    Mat img = imread("vanilia_cookie.jpg", IMREAD_GRAYSCALE);

    int newHeight = img.rows / 2;
    int newWidth = img.cols / 2;

        resize(img, img, Size(newWidth, newHeight));
    imwrite("resized_stud.png", img);

    Mat mask;
    threshold(img, mask, 220, 255, THRESH_BINARY_INV);
    imwrite("mask_stud.png", mask);

    Mat result;
    img.copyTo(result, mask);
    imwrite("result_stud.png", result);

    return 0;
}
```