Image Template Matching

Gohur Ali
Computer Vision

December 3, 2019

Contents

1 Problem 1 **2**
1.1 Algorithm 2

2 Problem 2 **3**
2.1 Code 3
2.2 Results 3
2.3 Complexity 3
2.4 Comparing to Convolution function 3
2.5 Greyscale Image Comparison 4

1 Problem 1

1.1 Algorithm

```cpp
std::vector<int> Transformer::average_BGR(cv::Mat& img,std::vector<int> bgr_totals) {
    float total_r = 0.0;
    float total_g = 0.0;
    float total_b = 0.0;
    for (int row = 0; row < img.rows; row++) {
        for (int col = 0; col < img.cols; col++) {
            int current_b = img.at<cv::Vec3b>(row, col)[0];
            int current_g = img.at<cv::Vec3b>(row, col)[1];
            int current_r = img.at<cv::Vec3b>(row, col)[2];
            total_b += current_b;
            total_g += current_g;
            total_r += current_r;
        }
    }
    int ave_b = static_cast<int>(total_b / bgr_totals[0]);
    int ave_g = static_cast<int>(total_g / bgr_totals[1]);
    int ave_r = static_cast<int>(total_r / bgr_totals[2]);
    std::vector<int> averages = { ave_b,ave_g,ave_r};
    return averages;
}
int Transformer::enhance_color(int current_color, int average_color, float factor) {
    float diff = (current_color - average_color) * factor;
    float new_val = average_color + diff;
    int new_val_rounded = static_cast<int>(new_val);
    new_val_rounded = new_val_rounded < 0 ? 0 : new_val_rounded;
    new_val_rounded = new_val_rounded > 255 ? 255 : new_val_rounded;
    return new_val_rounded;
}
cv::Mat Transformer::enhance_contrast(cv::Mat& img,std::vector<int>averages) {
    for (int row = 0; row < img.rows; row++) {
        for (int col = 0; col < img.cols; col++) {
            int current_b = img.at<cv::Vec3b>(row, col)[0];
            int current_g = img.at<cv::Vec3b>(row, col)[1];
            int current_r = img.at<cv::Vec3b>(row, col)[2];
            int new_b = this->enhance_color(current_b, averages[0], 1.5);
            int new_g = this->enhance_color(current_g, averages[1], 1.5);
            int new_r = this->enhance_color(current_r, averages[2], 1.5);
            img.at<cv::Vec3b>(row, col)[0] = new_b;
            img.at<cv::Vec3b>(row, col)[1] = new_g;
            img.at<cv::Vec3b>(row, col)[2] = new_r;
        }
    }
    return img;
}
```

Figure 1: Input image

Figure 2: Output image

```cpp
int main() {
    cv::Mat img = cv::imread("background.jpg");

    Transformer tf;
    std::vector<int> total_bgr = {img.rows * img.cols, img.rows * img.cols, img.rows * img.cols };
    printf("B: %d -- G: %d -- R: %d\n",total_bgr[0],total_bgr[1],total_bgr[2]);
    std::vector<int> ave_bgr = tf.average_BGR(img, total_bgr);
    printf("B: %d -- G: %d -- R: %d\n", ave_bgr[0], ave_bgr[1], ave_bgr[2]);
    cv::Mat enhanced_img = tf.enhance_contrast(img, ave_bgr);
    cv::imwrite("prob1_output.jpg", enhanced_img);
```

2 Problem 2

2.1 Code

```cpp
std::pair<std::vector<int>,cv::Mat> Transformer::edge_compare(cv::Mat& template_im, cv::Mat& search_im) {
    cv::Mat match_sum_img(search_im.rows, search_im.cols,CV_32S,cv::Scalar(0));
    std::vector<int> match_counts;
    for (int s_row = 0; s_row < search_im.rows - template_im.rows; s_row++) {
        for (int s_col = 0; s_col < search_im.cols - template_im.cols; s_col++) {
            int t_rows = template_im.rows;
            int t_cols = template_im.cols;
            int total_matches = 0;
            int s_row_over = s_row;
            int s_col_over = s_col;
            for (int t_row = 0, s_row_over = s_row; t_row < t_rows; t_row++, s_row_over++) {
                for (int t_col = 0, s_col_over = s_col; t_col < t_cols; t_col++, s_col_over++) {
                    int curr_s_pxl = search_im.at<uchar>(s_row_over, s_col_over);
                    int curr_t_pxl = template_im.at<uchar>(t_row, t_col);
                    if (curr_s_pxl == 255 && curr_t_pxl == 255) {
                        total_matches++;
                    }
                }
            }
            match_counts.push_back(total_matches);
            match_sum_img.at<int>(s_row, s_col) = total_matches;
        }
    }
    std::pair<std::vector<int>, cv::Mat> output = { match_counts,match_sum_img };
    return output;
}
```

```cpp
cv::Mat Transformer::show_match(cv::Mat& search_im, cv::Mat& template_im, std::pair<std::vector<int>, cv::Mat>& info) {
    // Get the maximum count
    int max_match_count = *std::max_element(info.first.begin(), info.first.end());
    int m_row = 0;
    int m_col = 0;
    cv::Mat output(search_im.rows, search_im.cols, CV_8UC3,cv::Scalar(0));
    for (int s_row = 0; s_row < search_im.rows; s_row++) {
        for (int s_col = 0; s_col < search_im.cols; s_col++) {
            int curr_count = info.second.at<int>(s_row, s_col);
            if (curr_count == max_match_count) {
                // Draw contents of whats in search im
                for (int si_row = s_row; si_row < s_row + template_im.rows; si_row++) {
                    for (int si_col = s_col; si_col < s_col + template_im.cols; si_col++) {
                        output.at<cv::Vec3b>(si_row, si_col)[2] = static_cast<uchar>(search_im.at<uchar>(si_row, si_col));
                    }
                }
                // Draw contents of what in the template im
                for (int si_row = s_row, t_row = 0; t_row < template_im.rows; si_row++,t_row++) {
                    for (int si_col = s_col, t_col = 0; t_col < template_im.cols; si_col++,t_col++) {
                        output.at<cv::Vec3b>(si_row, si_col)[1] = static_cast<uchar>(template_im.at<uchar>(t_row, t_col));
                    }
                }
            }
            if (s_col % 30) {
                m_col += 30;
            }
        }
        if (s_row % 30) {
            m_row += 30;
        }
    }
    return output;
}
```

2.2 Results

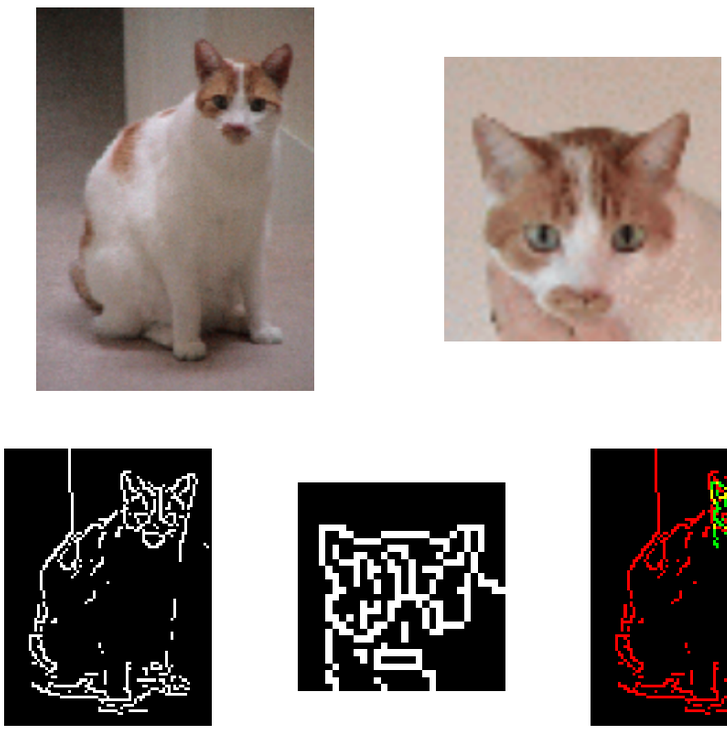

Figure 3: Resized and matched using sliding window approach

Using a sliding window method and calculating edge match counts, we can see that we find the location of the cat head in the search image. On the right most image, the red outline represents these search image, green being the template image, and yellow being direct matches.

2.3 Complexity

The complexity would be $O(n^2 \cdot m^2)$. This is because we first iterate through the search image which is and an $n \times n$ operation. Then within that, we iterate through the template image and calculate matching edges which is an $m \times m$ operation.

2.4 Comparing to Convolution function

The implementation shown above is similar to the convolution function. Rather it is like a sliding window approach where the sliding window is the template image over the search image. As the windows slides around the image, and any edges are overlapping are added to a match counter for each stride of the window. However, on the other hand, the convolution function performs a dot product with every stride. This would not perform the task needed where we need to determine how many edges overlap with eachother.

2.5 Greyscale Image Comparison

In order to compare greyscale images, one would need to compute pixel distances for position and intensity. In terms of this program, we would still utilize a sliding window approach, however, for each pixel in the window compute the distance for pixels with the same intesity and its position. If they are within a certain threshold it will be matched.